ESP32 Technical Reference Manual

Espressif Systems

December 23, 2016

About This Manual

ESP32 Technical Reference Manual is addressed to application developers. The manual provides detailed and complete information on how to use the ESP32 memory and peripherals.

For pin definition, electrical characteristics and package information, please see the ESP32 Datasheet.

Related Resources

Additional documentation and other resources about ESP32 can be accessed here: ESP32 Resources.

Release Notes

Date	Version	Release notes
2016.08	V1.0	Initial release.
2016.09	V1.1	Added Chapter I2C Controller.
2016.11	V1.2	Added Chapter PID/MPU/MMU; Updated Section IO_MUX and GPIO Matrix Register Summary; Updated Section LED_PWM Register Summary.
2016.12	V1.3	Added Chapter eFuse Controller; Added Chapter RSA Accelerator; Added Chapter Random Number Generator; Updated Section I2C Controller Interrupt and Section I2C Controller Registers.

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice. THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to the use of information in this document, is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein. The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Contents

4 IO_MUX and GPIO Matrix 31

5 I2C Controller 68

6 LED_PWM 89

15 Random Number Generator 175

16 PID/MPU/MMU 176

List of Tables

List of Figures

1. System and Memory

1.1 Introduction

The ESP32 is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. All embedded memory, external memory and peripherals are located on the data bus and/or the instruction bus of these CPUs.

With some minor exceptions (see below), the address mapping of two CPUs is symmetric, meaning that they use the same addresses to access the same memory. Multiple peripherals in the system can access embedded memory via DMA.

The two CPUs are named "PRO_CPU" and "APP_CPU" (for "protocol" and "application"), however, for most purposes the two CPUs are interchangeable.

1.2 Features

- Address Space
    - Symmetric address mapping
    - 4 GB (32-bit) address space for both data bus and instruction bus
    - 1296 KB embedded memory address space
    - 19704 KB external memory address space
    - 512 KB peripheral address space
    - Some embedded and external memory regions can be accessed by either data bus or instruction bus
    - 328 KB DMA address space
- Embedded Memory
    - 448 KB Internal ROM
    - 520 KB Internal SRAM
    - 8 KB RTC FAST Memory
    - 8 KB RTC SLOW Memory
- External Memory
  Off-chip SPI memory can be mapped into the available address space as external memory. Parts of the embedded memory can be used as transparent cache for this external memory.
    - Supports up to 16 MB off-Chip SPI Flash.
    - Supports up to 8 MB off-Chip SPI SRAM.
- Peripherals
    - 41 peripherals
- DMA
    - 13 modules are capable of DMA operation

The block diagram in Figure 1 illustrates the system structure, and the block diagram in Figure 2 illustrates the address map structure.

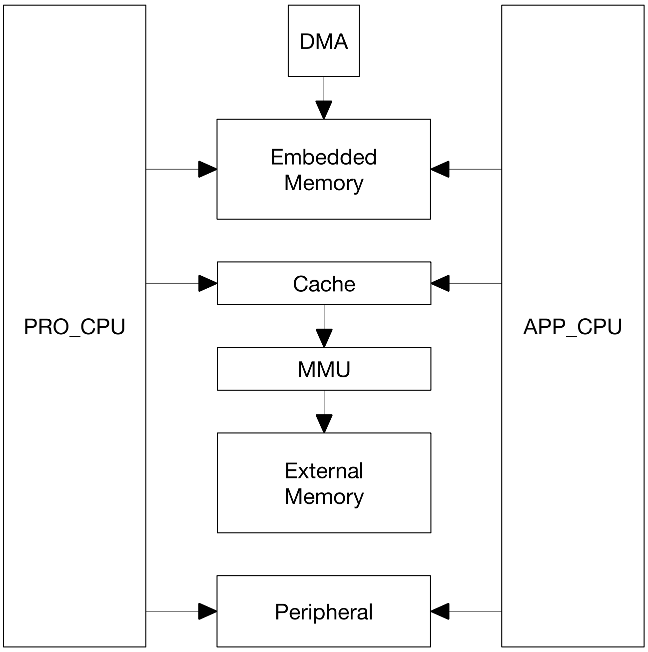


Figure 1: System Structure

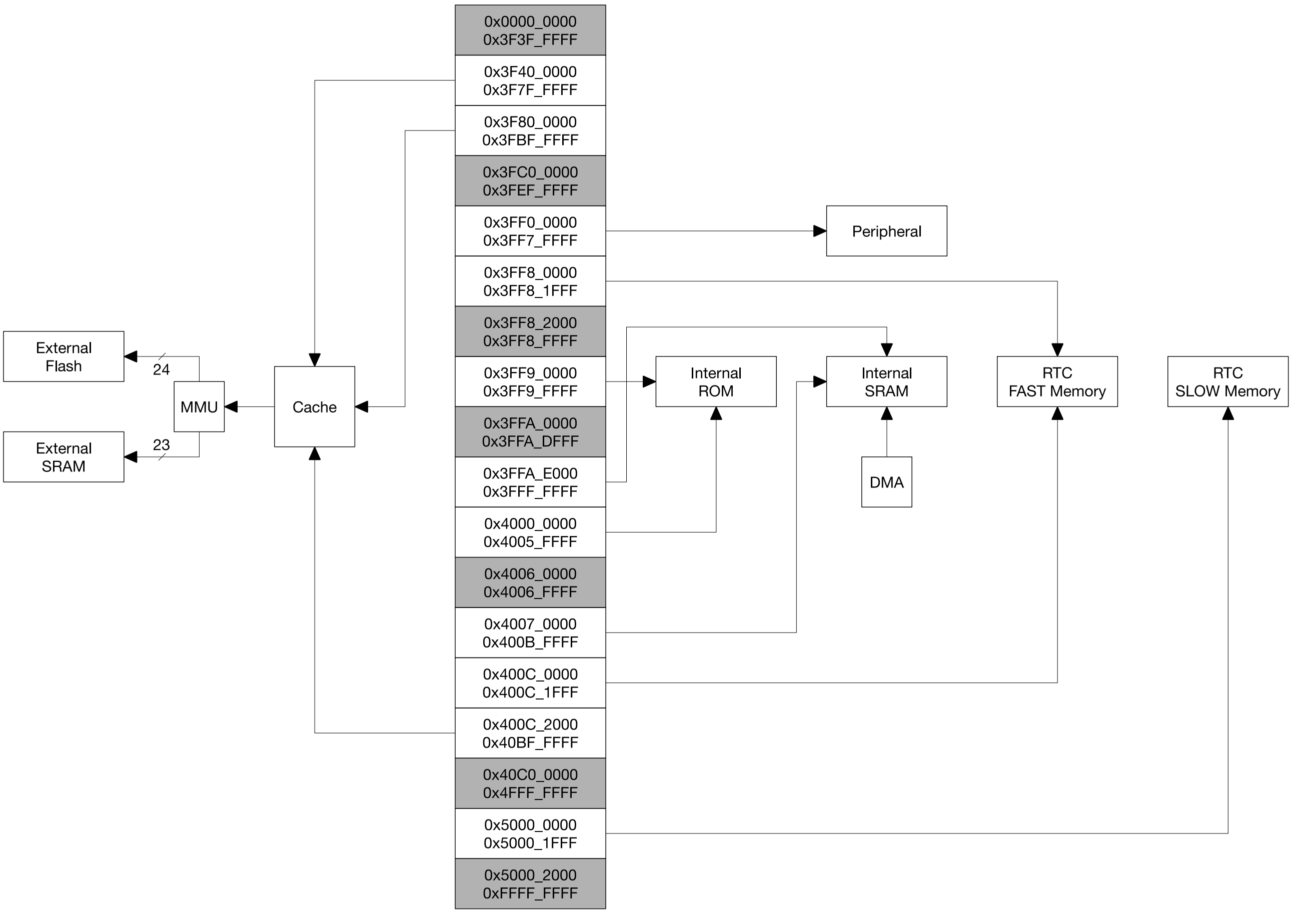


Figure 2: System Address Mapping

1.3 Functional Description

1.3.1 Address Mapping

Each of the two Harvard Architecture Xtensa LX6 CPUs has 4 GB (32-bit) address space. Address spaces are symmetric between the two CPUs.

Addresses below 0x4000_0000 are serviced using the data bus. Addresses in the range 0x4000_0000 ~ 0x4FFF_FFFF are serviced using the instruction bus. Finally, addresses over and including 0x5000_0000 are shared by the data and instruction bus.

The data bus and instruction bus are both little-endian: for example, byte addresses 0x0, 0x1, 0x2, 0x3 access the least significant, second least significant, second most significant, and the most significant bytes of the 32-bit word stored at the 0x0 address, respectively. The CPU can access data bus addresses via aligned or non-aligned byte, half-word and word read-and-write operations. The CPU can read and write data through the instruction bus, but only in a **word aligned manner**; non-word-aligned access will cause a CPU exception.

Each CPU can directly access embedded memory through both the data bus and the instruction bus, external memory which is mapped into the address space (via transparent caching & MMU), and peripherals. Table 1 illustrates address ranges that can be accessed by each CPU's data bus and instruction bus.

Some embedded memories and some external memories can be accessed via the data bus or the instruction bus. In these cases, the same memory is available to either of the CPUs at two address ranges.

Table 1: Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3F3F_FFFF		Reserved
Data	0x3F40_0000	0x3F7F_FFFF	4 MB	External Memory
Data	0x3F80_0000	0x3FBF_FFFF	4 MB	External Memory
	0x3FC0_0000	0x3FEF_FFFF	3 MB	Reserved
Data	0x3FF0_0000	0x3FF7_FFFF	512 KB	Peripheral
Data	0x3FF8_0000	0x3FFF_FFFF	512 KB	Embedded Memory
Instruction	0x4000_0000	0x400C_1FFF	776 KB	Embedded Memory
Instruction	0x400C_2000	0x40BF_FFFF	11512 KB	External Memory
	0x40C0_0000	0x4FFF_FFFF	244 MB	Reserved
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	Embedded Memory
	0x5000_2000	0xFFFF_FFFF		Reserved

1.3.2 Embedded Memory

The Embedded Memory consists of four segments: internal ROM (448 KB), internal SRAM (520 KB), RTC FAST memory (8 KB) and RTC SLOW memory (8 KB).

The 448 KB internal ROM is divided into two parts: Internal ROM 0 (384 KB) and Internal ROM 1 (64 KB).

The 520 KB internal SRAM is divided into three parts: Internal SRAM 0 (192 KB), Internal SRAM 1 (128 KB), and Internal SRAM 2 (200 KB).

RTC FAST Memory and RTC SLOW Memory are both implemented as SRAM.

Table 2 lists all embedded memories and their address ranges on the data and instruction buses.

Table 2: Embedded Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	Reserved	-
Data	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	-
	0x3FFA_0000	0x3FFA_DFFF	56 KB	Reserved	-
Data	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
Data	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
Instruction	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	-
	0x4006_0000	0x4006_FFFF	64 KB	Reserved	-
Instruction	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
Instruction	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	-
Instruction	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	-
Instruction	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
Instruction	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	-
Instruction	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	-

1.3.2.1 Internal ROM 0

The capacity of Internal ROM 0 is 384 KB. It is accessible by both CPUs through the address range 0x4000_0000 ~ 0x4005_FFFF, which is on the instruction bus.

The address range of the first 32 KB of the ROM 0 (0x4000_0000 ~ 0x4000_7FFF) can be remapped in order to access a part of Internal SRAM 1 that normally resides in a memory range of 0x400B_0000 ~ 0x400B_7FFF. While remapping, the 32 KB SRAM cannot be accessed by an address range of 0x400B_0000 ~ 0x400B_7FFF any more, but it can still be accessible through the data bus (0x3FFE_8000 ~ 0x3FFE_FFFF). This can be done on a per-CPU basis: setting bit 0 of register DPORT_PRO_BOOT_REMAP_CTRL_REG or DPORT_APP_BOOT_REMAP_CTRL_REG will remap SRAM for the PRO_CPU and APP_CPU, respectively.

1.3.2.2 Internal ROM 1

The capacity of Internal ROM 1 is 64 KB. It can be read by either CPU at an address range 0x3FF9_0000 ~ 0x3FF9_FFFF of the data bus.

1.3.2.3 Internal SRAM 0

The capacity of Internal SRAM 0 is 192 KB. Hardware can be configured to use the first 64KB to cache external memory access. When not used as cache, the first 64KB can be read and written by either CPU at addresses 0x4007_0000 ~ 0x4007_7FFF of the instruction bus. The remaining 128 KB can always be read and written by either CPU at addresses 0x4007_8000 ~ 0x4007_FFFF of instruction bus.

1.3.2.4 Internal SRAM 1

The capacity of Internal SRAM 1 is 128 KB. Either CPU can read and write this memory at addresses 0x3FFE_0000 ~ 0x3FFF_FFFF of the data bus, and also at addresses 0x400A_0000 ~ 0x400B_FFFF of the instruction bus.

The address range accessed via the instruction bus is in reverse order (word-wise) compared to access via the data bus. That is to say, address
0x3FFE_0000 and 0x400B_FFFC access the same word
0x3FFE_0004 and 0x400B_FFF8 access the same word
0x3FFE_0008 and 0x400B_FFF4 access the same word
……
0x3FFF_FFF4 and 0x400A_0008 access the same word
0x3FFF_FFF8 and 0x400A_0004 access the same word
0x3FFF_FFFC and 0x400A_0000 access the same word

The data bus and instruction bus of the CPU are still both little-endian, so the byte order of individual words is not reversed between address spaces. For example, address
0x3FFE_0000 accesses the least significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0001 accesses the second least significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0002 accesses the second most significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0003 accesses the most significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0004 accesses the least significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0005 accesses the second least significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0006 accesses the second most significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0007 accesses the most significant byte in the word accessed by 0x400B_FFF8.
……
0x3FFF_FFF8 accesses the least significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFF9 accesses the second least significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFA accesses the second most significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFB accesses the most significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFC accesses the least significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFD accesses the second most significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFE accesses the second most significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFF accesses the most significant byte in the word accessed by 0x400A_0000.

Part of this memory can be remapped onto the ROM 0 address space. See Internal Rom 0 for more information.

1.3.2.5 Internal SRAM 2

The capacity of Internal SRAM 2 is 200 KB. It can be read and written by either CPU at addresses 0x3FFA_E000 ~ 0x3FFD_FFFF on the data bus.

1.3.2.6 DMA

DMA uses the same addressing as the CPU data bus to read and write Internal SRAM 1 and Internal SRAM 2. This means DMA uses an address range of 0x3FFE_0000 ~ 0x3FFF_FFFF to read and write Internal SRAM 1 and an address range of 0x3FFA_E000 ~ 0x3FFD_FFFF to read and write Internal SRAM 2.

In the ESP32, 13 peripherals are equipped with DMA. Table 3 lists these peripherals.

Table 3: Module with DMA

<table>
<tr><td>UART0</td><td colspan="2">UART1</td><td>UART2</td></tr>
<tr><td>SPI1</td><td colspan="2">SPI2</td><td>SPI3</td></tr>
<tr><td colspan="2">I2S0</td><td colspan="2">I2S1</td></tr>
<tr><td colspan="2">SDIO Slave</td><td colspan="2">SDMMC</td></tr>
<tr><td colspan="4">EMAC</td></tr>
<tr><td colspan="2">BT</td><td colspan="2">WIFI</td></tr>
</table>

1.3.2.7 RTC FAST Memory

RTC FAST Memory is 8 KB of SRAM. It can be read and written by PRO_CPU only at an address range of 0x3FF8_0000 ~ 0x3FF8_1FFF on the data bus or at an address range of 0x400C_0000 ~ 0x400C_1FFF on the instruction bus. Unlike most other memory regions, RTC FAST memory cannot be accessed by the APP_CPU.

The two address ranges of PRO_CPU access RTC FAST Memory in the same order, so, for example, addresses 0x3FF8_0000 and 0x400C_0000 access the same word. **On the APP_CPU, these address ranges do not provide access to RTC FAST Memory or any other memory location.**

1.3.2.8 RTC SLOW Memory

RTC SLOW Memory is 8 KB of SRAM which can be read and written by either CPU at an address range of 0x5000_0000 ~ 0x5000_1FFF. This address range is shared by both the data bus and the instruction bus.

1.3.3 External Memory

The ESP32 can access external SPI flash and SPI SRAM as external memory. Table 4 provides a list of external memories that can be accessed by either CPU at a range of addresses on the data and instruction buses. When a CPU accesses external memory through the Cache and MMU, the cache will map the CPU's address to an external physical memory address (in the external memory's address space), according to the MMU settings. Due to this address mapping, the ESP32 can address up to 16 MB External Flash and 8 MB External SRAM.

Table 4: External Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3F40_0000	0x3F7F_FFFF	4 MB	External Flash	Read
Data	0x3F80_0000	0x3FBF_FFFF	4 MB	External SRAM	Read and Write
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x400C_2000	0x40BF_FFFF	11512 KB	External Flash	Read

1.3.4 Peripherals

The ESP32 has 41 peripherals. Table 5 specifically describes the peripherals and their respective address ranges. Nearly all peripheral modules can be accessed by either CPU at the same address with just a single exception; this being the PID Controller.

Table 5: Peripheral Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF0_0000	0x3FF0_0FFF	4 KB	DPort Register	
Data	0x3FF0_1000	0x3FF0_1FFF	4 KB	AES Accelerator	
Data	0x3FF0_2000	0x3FF0_2FFF	4 KB	RSA Accelerator	
Data	0x3FF0_3000	0x3FF0_3FFF	4 KB	SHA Accelerator	
Data	0x3FF0_4000	0x3FF0_4FFF	4 KB	Secure Boot	
	0x3FF0_5000	0x3FF0_FFFF	44 KB	Reserved	
Data	0x3FF1_0000	0x3FF1_3FFF	16 KB	Cache MMU Table	
	0x3FF1_4000	0x3FF1_EFFF	44 KB	Reserved	
Data	0x3FF1_F000	0x3FF1_FFFF	4 KB	PID Controller	Per-CPU peripheral
	0x3FF2_0000	0x3FF3_FFFF	128 KB	Reserved	
Data	0x3FF4_0000	0x3FF4_0FFF	4 KB	UART0	
	0x3FF4_1000	0x3FF4_1FFF	4 KB	Reserved	
Data	0x3FF4_2000	0x3FF4_2FFF	4 KB	SPI1	
Data	0x3FF4_3000	0x3FF4_3FFF	4 KB	SPI0	
Data	0x3FF4_4000	0x3FF4_4FFF	4 KB	GPIO	
	0x3FF4_5000	0x3FF4_7FFF	12 KB	Reserved	
Data	0x3FF4_8000	0x3FF4_8FFF	4 KB	RTC	
Data	0x3FF4_9000	0x3FF4_9FFF	4 KB	IO MUX	
	0x3FF4_A000	0x3FF4_AFFF	4 KB	Reserved	
Data	0x3FF4_B000	0x3FF4_BFFF	4 KB	SDIO Slave	One of three parts
Data	0x3FF4_C000	0x3FF4_CFFF	4 KB	UDMA1	
	0x3FF4_D000	0x3FF4_EFFF	8 KB	Reserved	
Data	0x3FF4_F000	0x3FF4_FFFF	4 KB	I2S0	
Data	0x3FF5_0000	0x3FF5_0FFF	4 KB	UART1	
	0x3FF5_1000	0x3FF5_2FFF	8 KB	Reserved	
Data	0x3FF5_3000	0x3FF5_3FFF	4 KB	I2C0	
Data	0x3FF5_4000	0x3FF5_4FFF	4 KB	UDMA0	

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF5_5000	0x3FF5_5FFF	4 KB	SDIO Slave	One of three parts
Data	0x3FF5_6000	0x3FF5_6FFF	4 KB	RMT	
Data	0x3FF5_7000	0x3FF5_7FFF	4 KB	PCNT	
Data	0x3FF5_8000	0x3FF5_8FFF	4 KB	SDIO Slave	One of three parts
Data	0x3FF5_9000	0x3FF5_9FFF	4 KB	LED PWM	
Data	0x3FF5_A000	0x3FF5_AFFF	4 KB	Efuse Controller	
Data	0x3FF5_B000	0x3FF5_BFFF	4 KB	Flash Encryption	
	0x3FF5_C000	0x3FF5_DFFF	8 KB	Reserved	
Data	0x3FF5_E000	0x3FF5_EFFF	4 KB	PWM0	
Data	0x3FF5_F000	0x3FF5_FFFF	4 KB	TIMG0	
Data	0x3FF6_0000	0x3FF6_0FFF	4 KB	TIMG1	
	0x3FF6_1000	0x3FF6_3FFF	12 KB	Reserved	
Data	0x3FF6_4000	0x3FF6_4FFF	4 KB	SPI2	
Data	0x3FF6_5000	0x3FF6_5FFF	4 KB	SPI3	
Data	0x3FF6_6000	0x3FF6_6FFF	4 KB	SYSCON	
Data	0x3FF6_7000	0x3FF6_7FFF	4 KB	I2C1	
Data	0x3FF6_8000	0x3FF6_8FFF	4 KB	SDMMC	
Data	0x3FF6_9000	0x3FF6_AFFF	8 KB	EMAC	
	0x3FF6_B000	0x3FF6_BFFF	4 KB	Reserved	
Data	0x3FF6_C000	0x3FF6_CFFF	4 KB	PWM1	
Data	0x3FF6_D000	0x3FF6_DFFF	4 KB	I2S1	
Data	0x3FF6_E000	0x3FF6_EFFF	4 KB	UART2	
Data	0x3FF6_F000	0x3FF6_FFFF	4 KB	PWM2	
Data	0x3FF7_0000	0x3FF7_0FFF	4 KB	PWM3	
	0x3FF7_1000	0x3FF7_4FFF	16 KB	Reserved	
Data	0x3FF7_5000	0x3FF7_5FFF	4 KB	RNG	
	0x3FF7_6000	0x3FF7_FFFF	40 KB	Reserved	

1.3.4.1 Asymmetric PID Controller Peripheral

There are two PID Controllers in the system. They serve the PRO_CPU and the APP_CPU, respectively. **The PRO_CPU and the APP_CPU can only access their own PID Controller and not that of their counterpart.** Each CPU uses the same memory range 0x3FF1_F000 ~ 3FF1_FFFF to access its own PID Controller.

1.3.4.2 Non-Contiguous Peripheral Memory Ranges

The SDIO Slave peripheral consists of three parts and the two CPUs use non-contiguous addresses to access these. The three parts are accessed at the address ranges 0x3FF4_B000 ~ 3FF4_BFFF, 0x3FF5_5000 ~ 3FF5_5FFF and 0x3FF5_8000 ~ 3FF5_8FFF of each CPU's data bus. Similarly to other peripherals, access to this peripheral is identical for both CPUs.

1.3.4.3 Memory Speed

The ROM as well as the SRAM are both clocked from CPU_CLK and can be accessed by the CPU in a single cycle. The RTC FAST memory is clocked from the APB_CLOCK and the RTC SLOW memory from the FAST_CLOCK, so access to these memories may be slower. DMA uses the APB_CLK to access memory.

Internally, the SRAM is organized in 32K-sized banks. Each CPU and DMA channel can simultaneously access the SRAM at full speed, provided they access addresses in different memory banks.

2. Interrupt Matrix

2.1 Introduction

The Interrupt Matrix embedded in the ESP32 independently allocates peripheral interrupt sources to the two CPUs' peripheral interrupts. This configuration is made to be highly flexible in order to meet many different needs.

2.2 Features

- Accepts 71 peripheral interrupt sources as input.
- Generates 26 peripheral interrupt sources per CPU as output (52 total).
- CPU NMI Interrupt Mask.
- Queries current interrupt status of peripheral interrupt sources.

The structure of the Interrupt Matrix is shown in Figure 3.

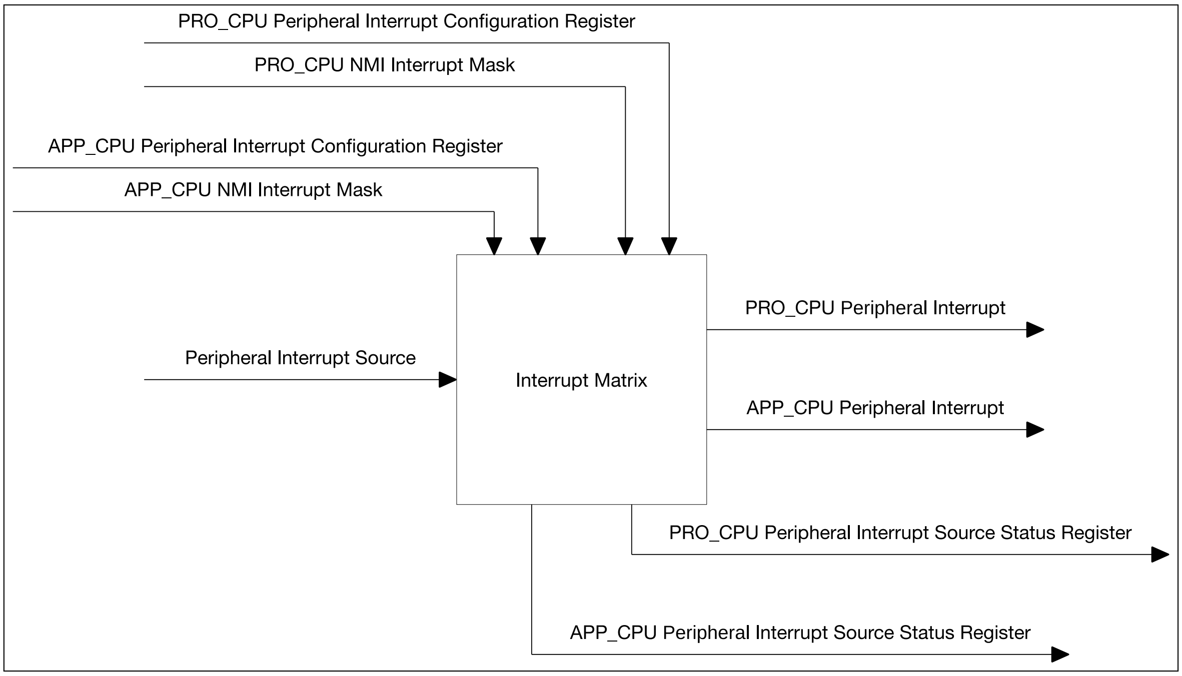


Figure 3: Interrupt Matrix Structure

2.3 Functional Description

2.3.1 Peripheral Interrupt Source

ESP32 has 71 peripheral interrupt sources in total. All peripheral interrupt sources are listed in table 6. 67 of 71 ESP32 peripheral interrupt sources can be allocated to either CPU.

The four remaining peripheral interrupt sources are CPU-specific, two per CPU. GPIO_INTERRUPT_PRO and GPIO_INTERRUPT_PRO_NMI can only be allocated to PRO_CPU. GPIO_INTERRUPT_APP and

GPIO_INTERRUPT_APP_NMI can only be allocated to APP_CPU. As a result, PRO_CPU and APP_CPU each have 69 peripheral interrupt sources.

Table 6: PRO_CPU, APP_CPU Interrupt Configuration

PRO_CPU					APP_CPU			
Peripheral Interrupt Configuration Register	Peripheral Interrupt Source							Peripheral Interrupt Configuration Register
	Bit	Status Register Name	No.	Name	No.	Status Register Name	Bit	
PRO_MAC_INTR_MAP_REG	0	PRO_INTR_STATUS_REG_0	0	MAC_INTR	0	APP_INTR_STATUS_REG_0	0	APP_MAC_INTR_MAP_REG
PRO_MAC_NMI_MAP_REG	1		1	MAC_NMI	1		1	APP_MAC_NMI_MAP_REG
PRO_BB_INT_MAP_REG	2		2	BB_INT	2		2	APP_BB_INT_MAP_REG
PRO_BT_MAC_INT_MAP_REG	3		3	BT_MAC_INT	3		3	APP_BT_MAC_INT_MAP_REG
PRO_BT_BB_INT_MAP_REG	4		4	BT_BB_INT	4		4	APP_BT_BB_INT_MAP_REG
PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5		5	APP_BT_BB_NMI_MAP_REG
PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6		6	APP_RWBT_IRQ_MAP_REG
PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5		5	APP_BT_BB_NMI_MAP_REG
PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6		6	APP_RWBT_IRQ_MAP_REG
PRO_RWBLE_IRQ_MAP_REG	7		7	RWBLE_IRQ	7		7	APP_RWBLE_IRQ_MAP_REG
PRO_RWBT_NMI_MAP_REG	8		8	RWBT_NMI	8		8	APP_RWBT_NMI_MAP_REG
PRO_RWBLE_NMI_MAP_REG	9		9	RWBLE_NMI	9		9	APP_RWBLE_NMI_MAP_REG
PRO_SLC0_INTR_MAP_REG	10		10	SLC0_INTR	10		10	APP_SLC0_INTR_MAP_REG
PRO_SLC1_INTR_MAP_REG	11		11	SLC1_INTR	11		11	APP_SLC1_INTR_MAP_REG
PRO_UHCI0_INTR_MAP_REG	12		12	UHCI0_INTR	12		12	APP_UHCI0_INTR_MAP_REG
PRO_UHCI1_INTR_MAP_REG	13		13	UHCI1_INTR	13		13	APP_UHCI1_INTR_MAP_REG
PRO_TG_T0_LEVEL_INT_MAP_REG	14		14	TG_T0_LEVEL_INT	14		14	APP_TG_T0_LEVEL_INT_MAP_REG
PRO_TG_T1_LEVEL_INT_MAP_REG	15		15	TG_T1_LEVEL_INT	15		15	APP_TG_T1_LEVEL_INT_MAP_REG
PRO_TG_WDT_LEVEL_INT_MAP_REG	16		16	TG_WDT_LEVEL_INT	16		16	APP_TG_WDT_LEVEL_INT_MAP_REG
PRO_TG_LACT_LEVEL_INT_MAP_REG	17		17	TG_LACT_LEVEL_INT	17		17	APP_TG_LACT_LEVEL_INT_MAP_REG
PRO_TG1_T0_LEVEL_INT_MAP_REG	18		18	TG1_T0_LEVEL_INT	18		18	APP_TG1_T0_LEVEL_INT_MAP_REG
PRO_TG1_T1_LEVEL_INT_MAP_REG	19		19	TG1_T1_LEVEL_INT	19		19	APP_TG1_T1_LEVEL_INT_MAP_REG
PRO_TG1_WDT_LEVEL_INT_MAP_REG	20		20	TG1_WDT_LEVEL_INT	20		20	APP_TG1_WDT_LEVEL_INT_MAP_REG
PRO_TG1_LACT_LEVEL_INT_MAP_REG	21		21	TG1_LACT_LEVEL_INT	21		21	APP_TG1_LACT_LEVEL_INT_MAP_REG
PRO_GPIO_INTERRUPT_PRO_MAP_REG	22		22	**GPIO_INTERRUPT_PRO** **GPIO_INTERRUPT_APP**	22		22	APP_GPIO_INTERRUPT_APP_MAP_REG
PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	23		23	**GPIO_INTERRUPT_PRO_NMI** **GPIO_INTERRUPT_APP_NMI**	23		23	APP_GPIO_INTERRUPT_APP_NMI_MAP_REG
PRO_CPU_INTR_FROM_CPU_0_MAP_REG	24		24	CPU_INTR_FROM_CPU_0	24		24	APP_CPU_INTR_FROM_CPU_0_MAP_REG
PRO_CPU_INTR_FROM_CPU_1_MAP_REG	25		25	CPU_INTR_FROM_CPU_1	25		25	APP_CPU_INTR_FROM_CPU_1_MAP_REG
PRO_CPU_INTR_FROM_CPU_2_MAP_REG	26		26	CPU_INTR_FROM_CPU_2	26		26	APP_CPU_INTR_FROM_CPU_2_MAP_REG
PRO_CPU_INTR_FROM_CPU_3_MAP_REG	27		27	CPU_INTR_FROM_CPU_3	27		27	APP_CPU_INTR_FROM_CPU_3_MAP_REG
PRO_SPI_INTR_0_MAP_REG	28		28	SPI_INTR_0	28		28	APP_SPI_INTR_0_MAP_REG
PRO_SPI_INTR_1_MAP_REG	29		29	SPI_INTR_1	29		29	APP_SPI_INTR_1_MAP_REG
PRO_SPI_INTR_2_MAP_REG	30		30	SPI_INTR_2	30		30	APP_SPI_INTR_2_MAP_REG
PRO_SPI_INTR_3_MAP_REG	31		31	SPI_INTR_3	31		31	APP_SPI_INTR_3_MAP_REG
PRO_I2S0_INT_MAP_REG	0	PRO_INTR_STATUS_REG_1	32	I2S0_INT	32	APP_INTR_STATUS_REG_1	0	APP_I2S0_INT_MAP_REG
PRO_I2S1_INT_MAP_REG	1		33	I2S1_INT	33		1	APP_I2S1_INT_MAP_REG
PRO_UART_INTR_MAP_REG	2		34	UART_INTR	34		2	APP_UART_INTR_MAP_REG
PRO_UART1_INTR_MAP_REG	3		35	UART1_INTR	35		3	APP_UART1_INTR_MAP_REG
PRO_UART2_INTR_MAP_REG	4		36	UART2_INTR	36		4	APP_UART2_INTR_MAP_REG
PRO_SDIO_HOST_INTERRUPT_MAP_REG	5		37	SDIO_HOST_INTERRUPT	37		5	APP_SDIO_HOST_INTERRUPT_MAP_REG
PRO_EMAC_INT_MAP_REG	6		38	EMAC_INT	38		6	APP_EMAC_INT_MAP_REG
PRO_PWM0_INTR_MAP_REG	7		39	PWM0_INTR	39		7	APP_PWM0_INTR_MAP_REG
PRO_PWM1_INTR_MAP_REG	8		40	PWM1_INTR	40		8	APP_PWM1_INTR_MAP_REG
PRO_PWM2_INTR_MAP_REG	9		41	PWM2_INTR	41		9	APP_PWM2_INTR_MAP_REG
PRO_PWM3_INTR_MAP_REG	10		42	PWM3_INTR	42		10	APP_PWM3_INTR_MAP_REG
PRO_LEDC_INT_MAP_REG	11		43	LEDC_INT	43		11	APP_LEDC_INT_MAP_REG
PRO_EFUSE_INT_MAP_REG	12		44	EFUSE_INT	44		12	APP_EFUSE_INT_MAP_REG
PRO_CAN_INT_MAP_REG	13		45	CAN_INT	45		13	APP_CAN_INT_MAP_REG
PRO_RTC_CORE_INTR_MAP_REG	14		46	RTC_CORE_INTR	46		14	APP_RTC_CORE_INTR_MAP_REG
PRO_RMT_INTR_MAP_REG	15		47	RMT_INTR	47		15	APP_RMT_INTR_MAP_REG
PRO_PCNT_INTR_MAP_REG	16		48	PCNT_INTR	48		16	APP_PCNT_INTR_MAP_REG
PRO_I2C_EXT0_INTR_MAP_REG	17		49	I2C_EXT0_INTR	49		17	APP_I2C_EXT0_INTR_MAP_REG
PRO_I2C_EXT1_INTR_MAP_REG	18		50	I2C_EXT1_INTR	50		18	APP_I2C_EXT1_INTR_MAP_REG
PRO_RSA_INTR_MAP_REG	19		51	RSA_INTR	51		19	APP_RSA_INTR_MAP_REG
PRO_SPI1_DMA_INT_MAP_REG	20		52	SPI1_DMA_INT	52		20	APP_SPI1_DMA_INT_MAP_REG

PRO_CPU					APP_CPU			
Peripheral Interrupt Configuration Register	Peripheral Interrupt Source							Peripheral Interrupt Configuration Register
	Status Register Bit	Status Register Name	No.	Name	No.	Status Register Name	Status Register Bit	
PRO_SPI2_DMA_INT_MAP_REG	21	PRO_INTR_STATUS_REG_1	53	SPI2_DMA_INT	53	APP_INTR_STATUS_REG_1	21	APP_SPI2_DMA_INT_MAP_REG
PRO_SPI3_DMA_INT_MAP_REG	22		54	SPI3_DMA_INT	54		22	APP_SPI3_DMA_INT_MAP_REG
PRO_WDG_INT_MAP_REG	23		55	WDG_INT	55		23	APP_WDG_INT_MAP_REG
PRO_TIMER_INT1_MAP_REG	24		56	TIMER_INT1	56		24	APP_TIMER_INT1_MAP_REG
PRO_TIMER_INT2_MAP_REG	25		57	TIMER_INT2	57		25	APP_TIMER_INT2_MAP_REG
PRO_TG_T0_EDGE_INT_MAP_REG	26		58	TG_T0_EDGE_INT	58		26	APP_TG_T0_EDGE_INT_MAP_REG
PRO_TG_T1_EDGE_INT_MAP_REG	27		59	TG_T1_EDGE_INT	59		27	APP_TG_T1_EDGE_INT_MAP_REG
PRO_TG_WDT_EDGE_INT_MAP_REG	28		60	TG_WDT_EDGE_INT	60		28	APP_TG_WDT_EDGE_INT_MAP_REG
PRO_TG_LACT_EDGE_INT_MAP_REG	29		61	TG_LACT_EDGE_INT	61		29	APP_TG_LACT_EDGE_INT_MAP_REG
PRO_TG1_T0_EDGE_INT_MAP_REG	30		62	TG1_T0_EDGE_INT	62		30	APP_TG1_T0_EDGE_INT_MAP_REG
PRO_TG1_T1_EDGE_INT_MAP_REG	31		63	TG1_T1_EDGE_INT	63		31	APP_TG1_T1_EDGE_INT_MAP_REG
PRO_TG1_WDT_EDGE_INT_MAP_REG	0	PRO_INTR_STATUS_REG_2	64	TG1_WDT_EDGE_INT	64	APP_INTR_STATUS_REG_2	0	APP_TG1_WDT_EDGE_INT_MAP_REG
PRO_TG1_LACT_EDGE_INT_MAP_REG	1		65	TG1_LACT_EDGE_INT	65		1	APP_TG1_LACT_EDGE_INT_MAP_REG
PRO_MMU_IA_INT_MAP_REG	2		66	MMU_IA_INT	66		2	APP_MMU_IA_INT_MAP_REG
PRO_MPU_IA_INT_MAP_REG	3		67	MPU_IA_INT	67		3	APP_MPU_IA_INT_MAP_REG
PRO_CACHE_IA_INT_MAP_REG	4		68	CACHE_IA_INT	68		4	APP_CACHE_IA_INT_MAP_REG

2.3.2 CPU Interrupt

Both of the two CPUs (PRO and APP) have 32 interrupts each, of which 26 are peripheral interrupts. All interrupts in a CPU are listed in Table 7.

Table 7: CPU Interrupts

No.	Category	Type	Priority Level
0	Peripheral	Level-Triggered	1
1	Peripheral	Level-Triggered	1
2	Peripheral	Level-Triggered	1
3	Peripheral	Level-Triggered	1
4	Peripheral	Level-Triggered	1
5	Peripheral	Level-Triggered	1
6	Internal	Timer.0	1
7	Internal	Software	1
8	Peripheral	Level-Triggered	1
9	Peripheral	Level-Triggered	1
10	Peripheral	Edge-Triggered	1
11	Internal	Profiling	3
12	Peripheral	Level-Triggered	1
13	Peripheral	Level-Triggered	1
14	Peripheral	NMI	NMI
15	Internal	Timer.1	3
16	Internal	Timer.2	5
17	Peripheral	Level-Triggered	1
18	Peripheral	Level-Triggered	1
19	Peripheral	Level-Triggered	2
20	Peripheral	Level-Triggered	2
21	Peripheral	Level-Triggered	2
22	Peripheral	Edge-Triggered	3
23	Peripheral	Level-Triggered	3
24	Peripheral	Level-Triggered	4
25	Peripheral	Level-Triggered	4
26	Peripheral	Level-Triggered	5
27	Peripheral	Level-Triggered	3
28	Peripheral	Edge-Triggered	4
29	Internal	Software	3
30	Peripheral	Edge-Triggered	4
31	Peripheral	Level-Triggered	5

2.3.3 Allocate Peripheral Interrupt Sources to Peripheral Interrupt on CPU

In this section:

- Source_X stands for any particular peripheral interrupt source.
- PRO_X_MAP_REG (or APP_X_MAP_REG) stands for any particular peripheral interrupt configuration

register of the PRO_CPU (or APP_CPU). The peripheral interrupt configuration register corresponds to the peripheral interrupt source Source_X. In Table 6 the registers listed under "PRO_CPU (APP_CPU) - Peripheral Interrupt Configuration Register" correspond to the peripheral interrupt sources listed in "Peripheral Interrupt Source - Name".

- Interrupt_P stands for CPU peripheral interrupt, numbered as Num_P. Num_P can take the ranges 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28, 30 ~ 31.
- Interrupt_I stands for the CPU internal interrupt numbered as Num_I. Num_I can take values 6, 7, 11, 15, 16, 29.

Using this terminology, the possible operations of the Interrupt Matrix controller can be described as follows:

- **Allocate peripheral interrupt source Source_X to CPU (PRO_CPU or APP_CPU)**
  Set PRO_X_MAP_REG or APP_X_MAP_REG to Num_P. Num_P can be any CPU peripheral interrupt number. CPU interrupts can be shared between multiple peripherals (see below).
- **Disable peripheral interrupt source Source_X for CPU (PRO_CPU or APP_CPU)**
  Set PRO_X_MAP_REG or APP_X _MAP_REG for peripheral interrupt source to any Num_I. The specific choice of internal interrupt number does not change behaviour, as none of the interrupt numbered as Num_I is connected to either CPU.
- **Allocate multiple peripheral sources Source_X*n* ORed to PRO_CPU (APP_CPU) peripheral interrupt**
  Set multiple PRO_X*n*_MAP_REG (APP_X*n*_MAP_REG) to the same Num_P. Any of these peripheral interrupts will trigger CPU Interrupt_P.

2.3.4 CPU NMI Interrupt Mask

The Interrupt Matrix temporarily masks all peripheral interrupt sources allocated to PRO_CPU's (or APP_CPU's) NMI interrupt, if it receives the signal PRO_CPU NMI Interrupt Mask (or APP_CPU NMI Interrupt Mask) from the peripheral PID Controller, respectively.

2.3.5 Query Current Interrupt Status of Peripheral Interrupt Source

The current interrupt status of a peripheral interrupt source can be read via the bit value in PRO_INTR_STATUS_REG_*n* (APP_INTR_STATUS_REG_*n*), as shown in the mapping in Table 6.

3. Reset and Clock

3.1 System Reset

3.1.1 Introduction

The ESP32 has three reset levels: CPU reset, Core reset, and System reset. None of these reset levels clear the RAM. Figure 4 shows the subsystems included in each reset level.

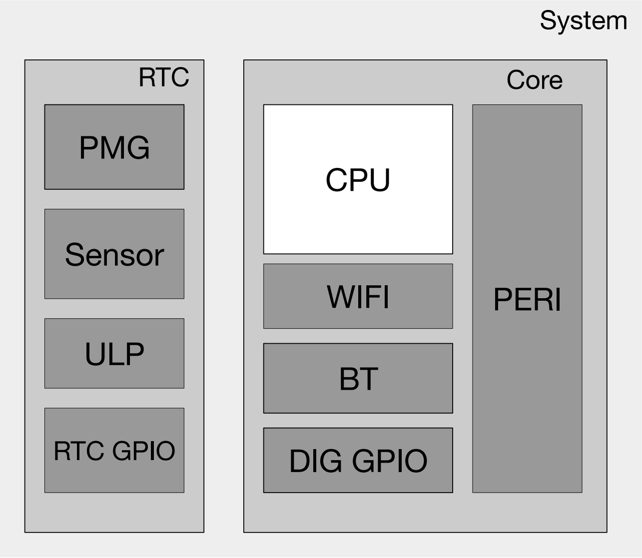


Figure 4: System Reset

- CPU reset: Only resets the registers of one or both of the CPU cores.
- Core reset: Resets all the digital registers, including CPU cores, external GPIO and digital GPIO. The RTC is not reset.
- System reset: Resets all the registers on the chip, including those of the RTC.

3.1.2 Reset Source

While most of the time the APP_CPU and PRO_CPU will be reset simultaneously, some reset sources are able to reset only one of the two cores. The reset reason for each core can be looked up individually: the PRO_CPU reset reason will be stored in RTC_CNTL_RESET_CAUSE_PROCPU, the reset reason for the APP_CPU in APP_CNTL_RESET_CAUSE_PROCPU. Table 8 shows the possible reset reason values that can be read from these registers.

Table 8: PRO_CPU and APP_CPU Reset Reason Values

PRO	APP	Source	Reset Type	Note
0x01	0x01	Chip Power On Reset	System Reset	-
0x10	0x10	RWDT System Reset	System Reset	See WDT Chapter.
0x0F	0x0F	Brown Out Reset	System Reset	See Power Management Chapter.
0x03	0x03	Software System Reset	Core Reset	Configure RTC_CNTL_SW_SYS_RST register.
0x05	0x05	Deep Sleep Reset	Core Reset	See Power Management Chapter.
0x07	0x07	MWDT0 Global Reset	Core Reset	See WDT Chapter.

PRO	APP	APP Source	Reset Type	Note
0x08	0x08	MWDT1 Global Reset	Core Reset	See WDT Chapter.
0x09	0x09	RWDT Core Reset	Core Reset	See WDT Chapter.
0x0B	-	MWDT0 CPU Reset	CPU Reset	See WDT Chapter.
0x0C	-	Software CPU Reset	CPU Reset	Configure RTC_CNTL_SW_APPCPU_RST register.
-	0x0B	MWDT1 CPU Reset	CPU Reset	See WDT Chapter.
-	0x0C	Software CPU Reset	CPU Reset	Configure RTC_CNTL_SW_APPCPU_RST register.
0x0D	0x0D	RWDT CPU Reset	CPU Reset	See WDT Chapter.
-	0xE	PRO CPU Reset	CPU Reset	Indicates that the PRO CPU has independently reset the APP CPU by configuring the DPORT_APPCPU_RESETTING register.

3.2 System Clock

3.2.1 Introduction

The ESP32 integrates multiple clock sources for the CPU cores, the peripherals and the RTC. These clocks can be configured to meet different requirements. Figure 5 shows the system clock structure.

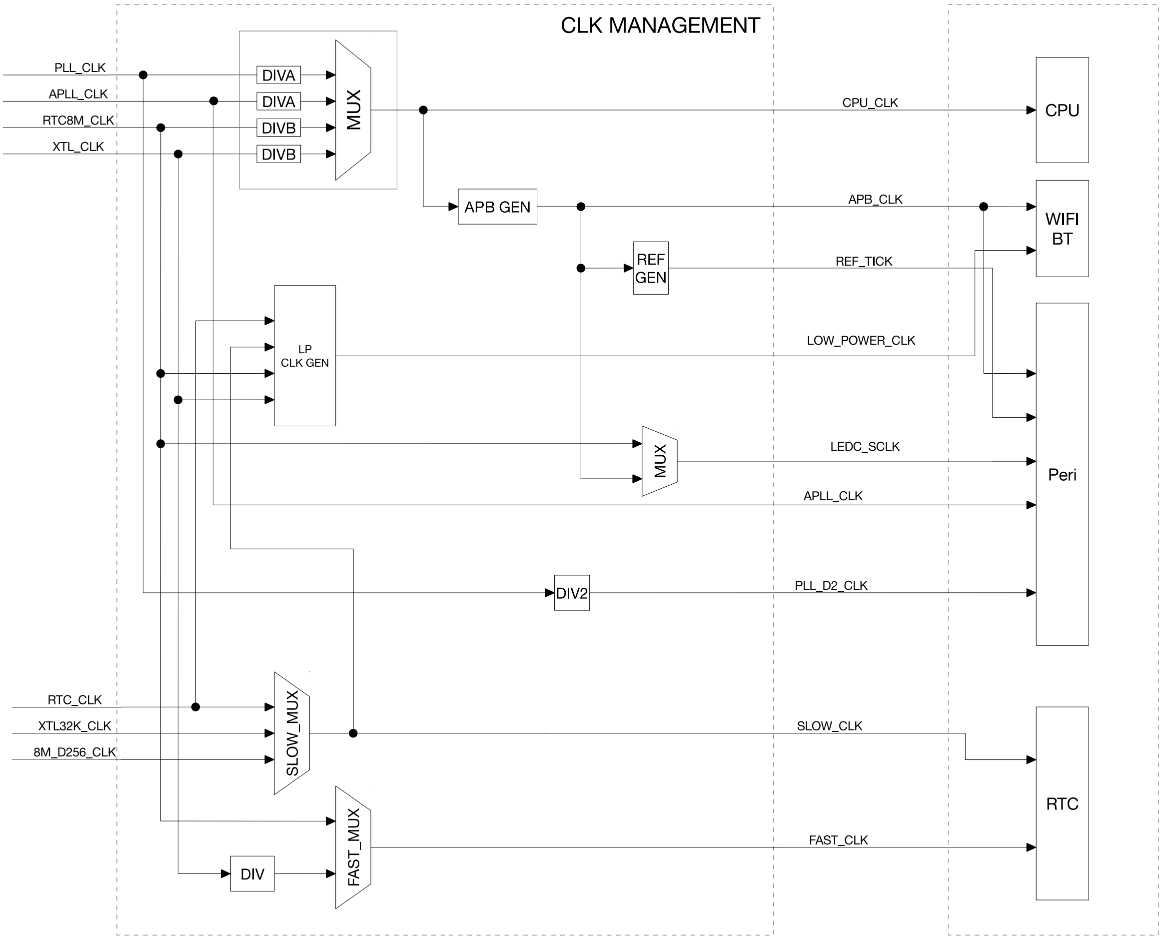


Figure 5: System Clock

3.2.2 Clock Source

The ESP32 can use an external crystal oscillator, an internal PLL or an oscillating circuit as a clock source. Specifically, the clock sources available are:

- High Speed Clocks
  - PLL_CLK is an internal PLL clock with a frequency of 320 MHz.
  - XTL_CLK is a clock signal generated using an external crystal with a frequency range of 2 ~ 40 MHz.
- Low Power Clocks
  - XTL32K_CLK is a clock generated using an external crystal with a frequency of 32 KHz.
  - RTC8M_CLK is an internal clock with a default frequency of 8 MHz. This frequency is adjustable.
  - RTC8M_D256_CLK is divided from RTC8M_CLK 256. Its frequency is (RTC8M_CLK / 256). With the default RTC8M_CLK frequency of 8 MHz, this clock runs at 31.250 KHz.
  - RTC_CLK is an internal low power clock with a default frequency of 150 KHz. This frequency is adjustable.
- Audio Clock
  - APLL_CLK is an internal Audio PLL clock with a frequency range of 16 ~ 128 MHz.

3.2.3 CPU Clock

As Figure 5 shows, CPU_CLK is the master clock for both CPU cores. CPU_CLK clock can be as high as 160 MHz when the CPU is in high performance mode. Alternatively, the CPU can run at lower frequencies to reduce power consumption.

The CPU_CLK clock source is determined by the RTC_CNTL_SOC_CLK_SEL register. PLL_CLK, APLL_CLK, RTC8M_CLK and XTL_CLK can be set as the CPU_CLK source; see Table 9 and 10.

Table 9: CPU_CLK Source

RTC_CNTL_SOC_CLK_SEL Value	Clock Source
0	XTL_CLK
1	PLL_CLK
2	RTC8M_CLK
3	APLL_CLK

Table 10: CPU_CLK Derivation

Clock Source	SEL*	CPU Clock
0 / XTL_CLK	-	CPU_CLK = XTL_CLK / (APB_CTRL_PRE_DIV_CNT+1) APB_CTRL_PRE_DIV_CNT range is 0 ~ 1023. Default is 0.
1 / PLL_CLK	0	CPU_CLK = PLL_CLK / 4 CPU_CLK frequency is 80 MHz
1 / PLL_CLK	1	CPU_CLK = PLL_CLK / 2 CPU_CLK frequency is 160 MHz
2 / RTC8M_CLK	-	CPU_CLK = RTC8M_CLK / (APB_CTRL_PRE_DIV_CNT+1) APB_CTRL_PRE_DIV_CNT range is 0 ~ 1023. Default is 0.
3 / APLL_CLK	0	CPU_CLK = APLL_CLK / 4
3 / APLL_CLK	1	CPU_CLK = APLL_CLK / 2

*SEL: DPORT_CPUPERIOD _SEL value

3.2.4 Peripheral Clock

Peripheral clocks include APB_CLK, REF_TICK, LEDC_SCLK, APLL_CLK and PLL_D2_CLK.
Table 11 shows which clocks can be used by which peripherals.

Table 11: Peripheral Clock Usage

Peripherals	APB_CLK	REF_TICK	LEDC_SCLK	APLL_CLK	PLL_D2_CLK
EMAC	Y	N	N	Y	N
TIMG	Y	N	N	N	N
I2S	Y	N	N	Y	Y
UART	Y	Y	N	N	N
RMT	Y	Y	N	N	N
LED PWM	Y	Y	Y	N	N
PWM	Y	N	N	N	N
I2C	Y	N	N	N	N
SPI	Y	N	N	N	N
PCNT	Y	N	N	N	N
Efuse Controller	Y	N	N	N	N
SDIO Slave	Y	N	N	N	N
SDMMC	Y	N	N	N	N

3.2.4.1 APB_CLK Source

The APB_CLK is derived from CPU_CLK as detailed in Table 12. The division factor depends on the CPU_CLK source.

Table 12: APB_CLK Derivation

CPU_CLK Source	APB_CLK
PLL_CLK	PLL_CLK / 4
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

3.2.4.2 REF_TICK Source

REF_TICK is derived from APB_CLK via a divider. The divider value used depends on the APB_CLK source, which in turn depends on the CPU_CLK source.

By configuring correct divider values for each APB_CLK source, the user can ensure that the REF_TICK frequency does not change when CPU_CLK changes source, causing the APB_CLK frequency to change.

Clock divider registers are shown in Table 13.

Table 13: REF_TICK Derivation

CPU_CLK & APB_CLK Source	Clock Divider Register
PLL_CLK	APB_CTRL_PLL_TICK_NUM
XTAL_CLK	APB_CTRL_XTAL_TICK_NUM
APLL_CLK	APB_CTRL_APLL_TICK_NUM
RTC8M_CLK	APB_CTRL_CK8M_TICK_NUM

3.2.4.3 LEDC_SCLK Source

The LEDC_SCLK clock source is selected by the LEDC_APB_CLK_SEL register, as shown in Table 14.

Table 14: LEDC_SCLK Derivation

LEDC_APB_CLK_SEL Value	LEDC_SCLK Source
1	RTC8M_CLK
0	APB_CLK

3.2.4.4 APLL_SCLK Source

The APLL_CLK is sourced from PLL_CLK, with its output frequency configured using the APLL configuration registers.

3.2.4.5 PLL_D2_CLK Source

PLL_D2_CLK is half the PLL_CLK frequency.

3.2.4.6 Clock Source Considerations

Most peripherals will operate using the APB_CLK frequency as a reference. When this frequency changes, the peripherals will need to update their clock configuration to operate at the same frequency after the change. Peripherals accessing REF_TICK can continue operating normally when switching clock sources, without changing clock source. Please see Table 11 for details.

The LED PWM module can use RTC8M_CLK as a clock source when APB_CLK is disabled. In other words, when the system is in low-power consumption mode (see power manager module), normal peripherals will be halted (APB_CLK is turned off), but the LED PWM can work normally via RTC8M_CLK.

3.2.5 Wi-Fi BT Clock

Wi-Fi and BT can only operate if APB_CLK uses PLL_CLK as its clock source. Suspending PLL_CLK requires Wi-Fi and BT to both have entered low-power consumption mode first.

For LOW_POWER_CLK, one of RTC_CLK, SLOW_CLK, RTC8M_CLK or XTL_CLK can be selected as the low-power consumption mode clock source for Wi-Fi and BT.

3.2.6 RTC Clock

The clock sources of SLOW_CLK and FAST_CLK are low-frequency clocks. The RTC module can operate when most other clocks are stopped.

SLOW_CLK is used to clock the Power Management module. It can be sourced from RTC_CLK, XTL32K_CLK or RTC8M_D256_CLK

FAST_CLK is used to clock the On-chip Sensor module. It can be sourced from a divided XTL_CLK or from RTC8M_CLK.

4. IO_MUX and GPIO Matrix

4.1 Introduction

The ESP32 chip features 40 physical GPIO pads. Some GPIO pads can neither be used nor have the corresponding pins on the chip package. Each pad can be used as a general-purpose I/O, or be connected to an internal peripheral signal. The IO_MUX, RTC IO_MUX and the GPIO matrix are responsible for routing signals from the peripherals to GPIO pads. Together these systems provide highly configurable I/O.

This chapter describes the signal selection and connection between the digital pads (FUNC_SEL, IE, OE, WPU, WDU, etc), 256 peripheral input/output signals (control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc), fast peripheral input/output signals (control signals: IE, OE, etc), and RTC IO_MUX.

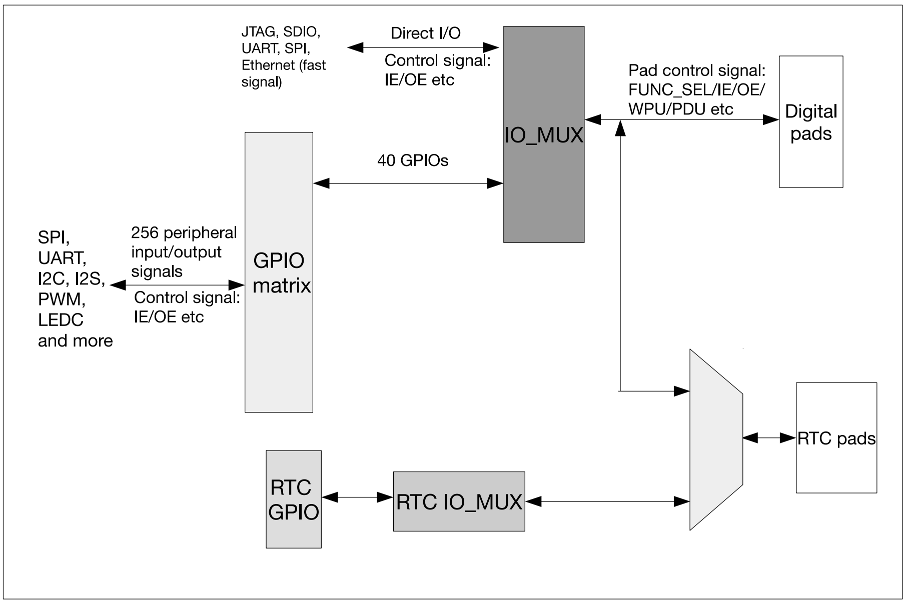


Figure 6: IO_MUX, RTC IO_MUX and GPIO Matrix Overview

1. The IO_MUX contains one register per GPIO pad. Each pad can be configured to perform a ”GPIO” function (when connected to the GPIO Matrix) or a direct function (bypassing the GPIO Matrix). Some high-speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency digital performance. In this case, the IO_MUX is used to connect these pads directly to the peripheral.)

   See Section 4.10 for a list of IO_MUX functions for each I/O pad.

2. The GPIO Matrix is a full-switching matrix between the peripheral input/output signals and the pads.

   - For input to the chip: Each of the 256 internal peripheral inputs can select any GPIO pad as the input source.
   - For output from the chip: The output signal of each of the 40 GPIO pads can be from one of the 256 peripheral output signals.

   See Section 4.9 for a list of GPIO Matrix peripheral signals.

3. RTC IO_MUX is used to connect GPIO pads to their low-power and analog functions. Only a subset of GPIO pads have these optional ”RTC” functions.

   See Section 4.11 for a list of RTC IO_MUX functions.

4.2 Peripheral Input via GPIO Matrix

4.2.1 Summary

To receive a peripheral input signal via the GPIO Matrix, the GPIO Matrix is configured to source the peripheral signal's input index (0-255) from one of the 40 GPIOs (0-39).

The input signal is read from the GPIO pad through the IO_MUX. The IO_MUX must be configured to set the chosen pad to "GPIO" function. This causes the GPIO pad input signal to be routed into the GPIO Matrix, which in turn routes it to the selected peripheral input.

4.2.2 Functional Description

Figure 7 shows the logic for input selection via GPIO Matrix.

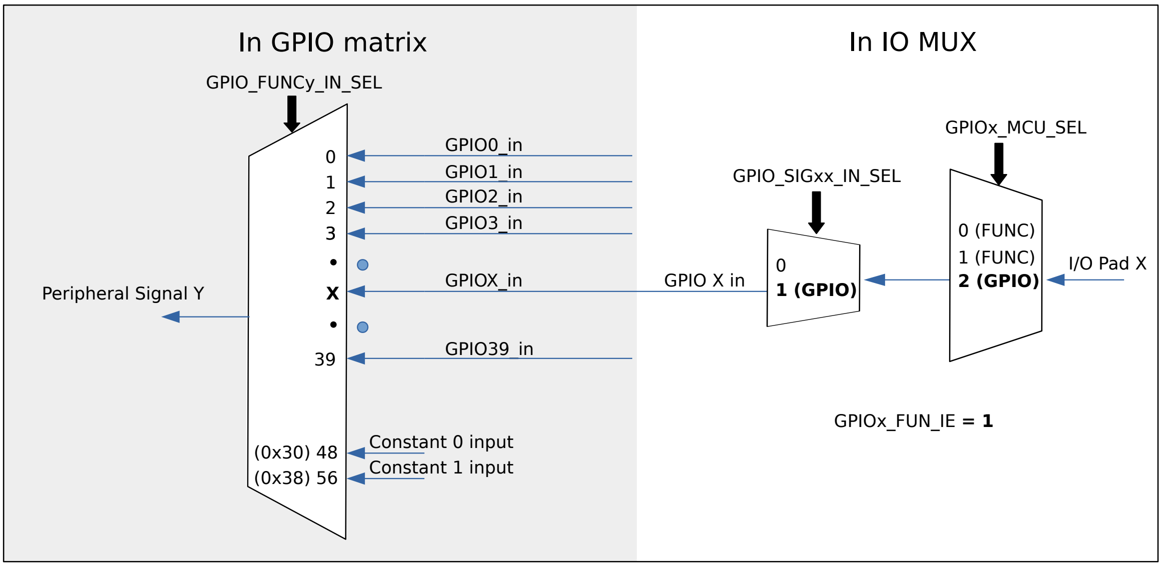


Figure 7: Peripheral Input via IO_MUX, GPIO Matrix

To read GPIO pad *X* into peripheral signal *Y*, follow the steps below:

1. Configure the GPIO_FUNC*y*_IN_SEL_CFG register for peripheral signal *Y* in the GPIO Matrix:
    - Set the GPIO_FUNC*x*_IN_SEL field to the number of the GPIO pad *X* to read from.
2. Configure the GPIO_FUNC*x*_OUT_SEL_CFG and GPIO_ENABLE_DATA[*x*] for GPIO pad *X* in the GPIO Matrix:
    - For input only signals, the pad output can be disabled by setting the GPIO_FUNC*x*_OEN_SEL bits to one and GPIO_ENABLE_DATA[*x*] to zero. Otherwise, there is no need to disable output.
3. Configure the IO_MUX register for GPIO pad *X*:
    - Set the function field to GPIO.
    - Enable the input by setting the *xx*_FUN_IE bit.
    - Set *xx*_FUN_WPU and *xx*_FUN_WPD fields, as desired, to enable internal pull-up/pull-down resistors.

Notes:

- One input pad can be connected to multiple input_signals.

- The input signal can be inverted with GPIO_FUNC*x*_IN_INV_SEL.
- It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pad. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
  - When GPIO_FUNC*x*_IN_SEL is 0x30, input_signal_*x* is always 0.
  - When GPIO_FUNC*x*_IN_SEL is 0x38, input_signal_*x* is always 1.

4.2.3 Simple GPIO Input

The GPIO_IN_DATA register holds the input values of each GPIO pad.

The input value of any GPIO pin can be read at any time without configuring the GPIO Matrix for a particular peripheral signal. However, it is necessary to configure the *xx*_FUN_IE register for pad *X*, as shown in Section 4.2.2.

4.3 Peripheral Output via GPIO Matrix

4.3.1 Summary

To output a signal from a peripheral via the GPIO Matrix, the GPIO Matrix is configured to route the peripheral output signal (0-255) to one of the first 34 GPIOs (0-33). (Note that GPIO pads 34-39 cannot be used as outputs.)

The output signal is routed from the peripheral into the GPIO Matrix. It is then routed into the IO_MUX, which is configured to set the chosen pad to ”GPIO” function. This causes the output GPIO signal to be connected to the pad.

4.3.2 Functional Description

One of 256 input signals can be selected to go through the GPIO matrix into the IO_MUX and then to a pad. Figure 8 illustrates the configuration.

To output peripheral signal *Y* to particular GPIO pad *X*, follow these steps:

1. Configure the GPIO_FUNC*x*_OUT_SEL_CFG register and GPIO_ENABLE_DATA[*x*] of GPIO *X* in the GPIO Matrix:
   - Set GPIO_FUNC*x*_OUT_SEL to the index of desired peripheral output signal *Y*.
   - Set the GPIO_FUNC*x*_OEN_SEL bits and GPIO_ENABLE_DATA[*x*] to enable output mode, or clear GPIO_FUNC*x*_OEN_SEL to zero so that the output enable signal will be decided by the internal logic function.
2. Alternatively, to enable open drain mode set the GPIO_PIN*x*_PAD_DRIVER bit in the GPIO_PIN*x* register.
3. Configure the I/O mux register for GPIO pad *X*:
   - Set the function field to GPIO.
   - Set the *xx*_FUN_DRV field to the desired value for output strength. The higher the value is, the stronger the output becomes. Pull up/down the pad by configuring *xx*_FUNC_WPU and *xx*_FUNC_WPD registers in open drain mode.

Notes:

- The output signal from a single peripheral can be sent to multiple pads simultaneously.
- Only the first 34 GPIOs (0-33) can be used as outputs.
- The output signal can be inverted by setting the GPIO_FUNC*x*_OUT_INV_SEL bit.

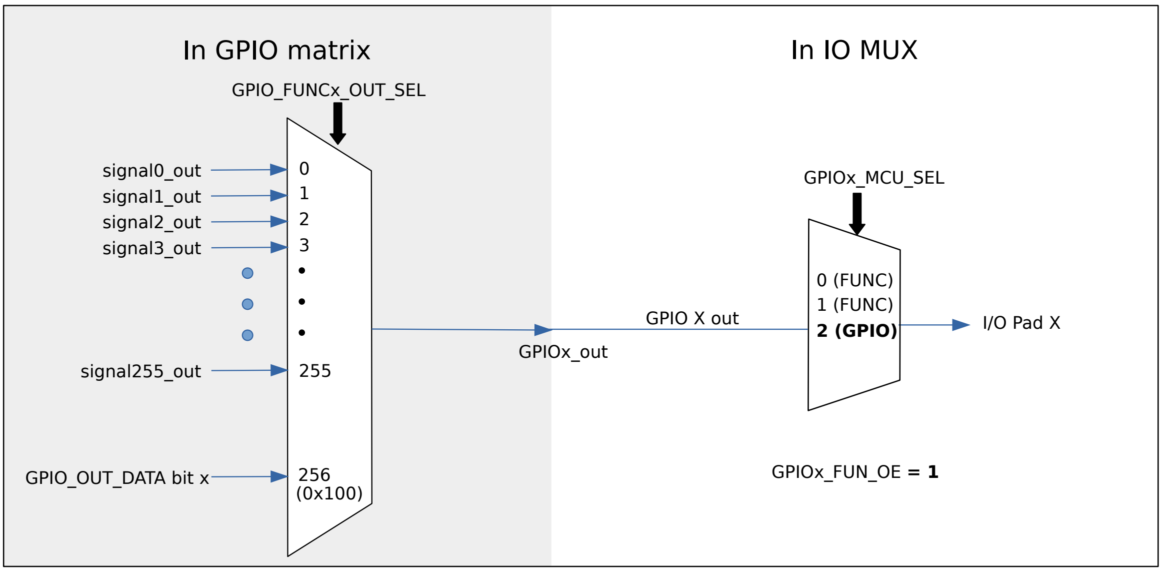


Figure 8: Output via GPIO Matrix

4.3.3 Simple GPIO Output

The GPIO Matrix can also be used for simple GPIO output - setting a bit in the GPIO_OUT_DATA register will write to the corresponding GPIO pad.

To configure a pad as simple GPIO output, the GPIO Matrix GPIO_FUNC*x*_OUT_SEL register is configured with a special peripheral index value (0x100).

4.4 Direct I/O via IO_MUX

4.4.1 Summary

Some high speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency digital performance. In this case, the IO_MUX is used to connect these pads directly to the peripheral.

Selecting this option is less flexible than using the GPIO Matrix, as the IO_MUX register for each GPIO pad can only select from a limited number of functions. However, better high-frequency digital performance will be maintained.

4.4.2 Functional Description

Two registers must be configured in order to bypass the GPIO Matrix for peripheral I/O:

1. IO_MUX for the GPIO pad must be set to the desired pad function (Section 4.10 has a list of pad functions).
2. For inputs, the SIG_IN_SEL register must be set to route the input directly to the peripheral.

4.5 RTC IO_MUX for Low Power and Analog I/O

4.5.1 Summary

18 pins have low power (RTC domain) capabilities and analog functions which are handled by the RTC subsystem of ESP32. The IO_MUX and GPIO Matrix are not used for these functions; rather, the RTC_MUX is used to redirect the I/O to the RTC subsystem.

When configured as RTC GPIOs, the output pads can still retain the output level value when the chip is in Deep-sleep mode, and the input pads can wake up the chip from Deep-sleep.

Section 4.11 has a list of RTC_MUX pins and their functions.

4.5.2 Functional Description

Each pad with analog and RTC functions is controlled by the RTC_IO_TOUCH_PAD*x*_TO_GPIO bit in the RTC_GPIO_PIN*x* register. By default this bit is set to 1, routing all I/O via the IO_MUX subsystem as described in earlier subsections.

If the RTC_IO_TOUCH_PAD*x*_TO_GPIO bit is cleared, then I/O to and from that pad is routed to the RTC subsystem. In this mode, the RTC_GPIO_PIN*x* register is used for digital I/O and the analog features of the pad are also available. See Section 4.11 for a list of RTC pin functions.

See 4.11 for a table mapping GPIO pads to their RTC equivalent pins and analog functions. Note that the RTC_IO_PIN*x* registers use the RTC GPIO pin numbering, not the GPIO pad numbering.

4.6 Light-sleep Mode Pin Functions

Pins can have different functions when the ESP32 is in Light-sleep mode. If the GPIO*xx*_SLP_SEL bit in the IO_MUX register for a GPIO pad is set to 1, a different set of registers is used to control the pad when the ESP32 is in Light-sleep mode:

Table 15: IO_MUX Light-sleep Pin Function Registers

IO_MUX Function	Normal Execution OR GPIO*xx*_SLP_SEL = 0	Light-sleep Mode AND GPIO*xx*_SLP_SEL = 1
Output Drive Strength	GPIO*xx*_FUNC_DRV	GPIO*xx*_MCU_DRV
Pullup Resistor	GPIO*xx*_FUNC_WPU	GPIO*xx*_MCU_WPU
Pulldown Resistor	GPIO*xx*_FUNC_WPD	GPIO*xx*_MCU_WPD
Output Enable	(From GPIO Matrix _OEN field)	GPIO*xx*_MCU_OE

If GPIO*xx*_SLP_SEL is set to 0, the pin functions remain the same in both normal execution and Light-sleep modes.

4.7 Pad Hold Feature

Each IO pad (including the RTC pads) has an individual hold function controlled by a RTC register. When the pad is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO_MUX configuration or GPIO configuration is modified. Users can use the hold function for the pads to retain the pad state through a core reset and system reset triggered by watchdog time-out or Deep-sleep events.

4.8 I/O Pad Power Supply

IO pad power supply is shown in Figure 9.

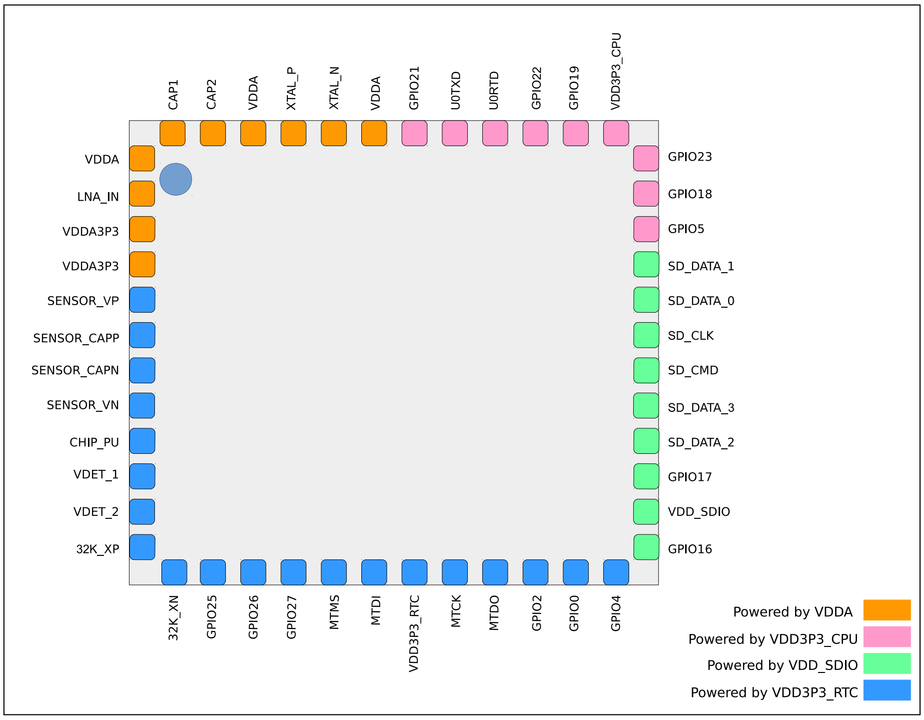


Figure 9: ESP32 I/O Pad Power Sources

- Pads marked blue are RTC pads that have their individual analog function and can also act as normal digital IO pads. For details, please see Section 4.11.
- Pads marked pink and green have digital functions only.
- Pads marked green can be powered externally or internally via VDD_SDIO (see below).

4.8.1 VDD_SDIO Power Domain

VDD_SDIO can source or sink current, allowing this power domain to be powered externally or internally. To power VDD_SDIO externally, apply the same power supply of VDD3P3_RTC to the VDD_SDIO pad.

Without an external power supply, the internal regulator will supply VDD_SDIO. The VDD_SDIO voltage can be configured to be either 1.8V or 3.3V (the same as that at VRTC), depending on the state of the MTDI pad at reset - a high level configures 1.8V and a low level configures 3.3V. Setting the efuse bit determines the default voltage of the VDD_SDIO. In addition, software can change the voltage of the VDD_SDIO by configuring register bits.

4.9 Peripheral Signal List

Table 16 contains a list of Peripheral Input/Output signals used by the GPIO Matrix:

Table 16: GPIO Matrix Peripheral Signals

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
0	SPICLK_in	SPICLK_out	YES
1	SPIQ_in	SPIQ_out	YES
2	SPID_in	SPID_out	YES
3	SPIHD_in	SPIHD_out	YES
4	SPIWP_in	SPIWP_out	YES
5	SPICS0_in	SPICS0_out	YES
6	SPICS1_in	SPICS1_out	
7	SPICS2_in	SPICS2_out	
8	HSPICLK_in	HSPICLK_out	YES
9	HSPIQ_in	HSPIQ_out	YES
10	HSPID_in	HSPID_out	YES
11	HSPICS0_in	HSPICS0_out	YES
12	HSPIHD_in	HSPIHD_out	YES
13	HSPIWP_in	HSPIWP_out	YES
14	U0RXD_in	U0TXD_out	YES
15	U0CTS_in	U0RTS_out	YES
16	U0DSR_in	U0DTR_out	
17	U1RXD_in	U1TXD_out	YES
18	U1CTS_in	U1RTS_out	YES
23	I2S0O_BCK_in	I2S0O_BCK_out	
24	I2S1O_BCK_in	I2S1O_BCK_out	
25	I2S0O_WS_in	I2S0O_WS_out	
26	I2S1O_WS_in	I2S1O_WS_out	
27	I2S0I_BCK_in	I2S0I_BCK_out	
28	I2S0I_WS_in	I2S0I_WS_out	
29	I2CEXT0_SCL_in	I2CEXT0_SCL_out	
30	I2CEXT0_SDA_in	I2CEXT0_SDA_out	
31	pwm0_sync0_in	sdio_tohost_int_out	
32	pwm0_sync1_in	pwm0_out0a	
33	pwm0_sync2_in	pwm0_out0b	
34	pwm0_f0_in	pwm0_out1a	
35	pwm0_f1_in	pwm0_out1b	
36	pwm0_f2_in	pwm0_out2a	
37		pwm0_out2b	
39	pcnt_sig_ch0_in0		
40	pcnt_sig_ch1_in0		
41	pcnt_ctrl_ch0_in0		
42	pcnt_ctrl_ch1_in0		
43	pcnt_sig_ch0_in1		
44	pcnt_sig_ch1_in1		
45	pcnt_ctrl_ch0_in1		
46	pcnt_ctrl_ch1_in1		
47	pcnt_sig_ch0_in2		

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
48	pcnt_sig_ch1_in2		
49	pcnt_ctrl_ch0_in2		
50	pcnt_ctrl_ch1_in2		
51	pcnt_sig_ch0_in3		
52	pcnt_sig_ch1_in3		
53	pcnt_ctrl_ch0_in3		
54	pcnt_ctrl_ch1_in3		
55	pcnt_sig_ch0_in4		
56	pcnt_sig_ch1_in4		
57	pcnt_ctrl_ch0_in4		
58	pcnt_ctrl_ch1_in4		
61	HSPICS1_in	HSPICS1_out	
62	HSPICS2_in	HSPICS2_out	
63	VSPICLK_in	VSPICLK_out_mux	YES
64	VSPIQ_in	VSPIQ_out	YES
65	VSPID_in	VSPID_out	YES
66	VSPIHD_in	VSPIHD_out	YES
67	VSPIWP_in	VSPIWP_out	YES
68	VSPICS0_in	VSPICS0_out	YES
69	VSPICS1_in	VSPICS1_out	
70	VSPICS2_in	VSPICS2_out	
71	pcnt_sig_ch0_in5	ledc_hs_sig_out0	
72	pcnt_sig_ch1_in5	ledc_hs_sig_out1	
73	pcnt_ctrl_ch0_in5	ledc_hs_sig_out2	
74	pcnt_ctrl_ch1_in5	ledc_hs_sig_out3	
75	pcnt_sig_ch0_in6	ledc_hs_sig_out4	
76	pcnt_sig_ch1_in6	ledc_hs_sig_out5	
77	pcnt_ctrl_ch0_in6	ledc_hs_sig_out6	
78	pcnt_ctrl_ch1_in6	ledc_hs_sig_out7	
79	pcnt_sig_ch0_in7	ledc_ls_sig_out0	
80	pcnt_sig_ch1_in7	ledc_ls_sig_out1	
81	pcnt_ctrl_ch0_in7	ledc_ls_sig_out2	
82	pcnt_ctrl_ch1_in7	ledc_ls_sig_out3	
83	rmt_sig_in0	ledc_ls_sig_out4	
84	rmt_sig_in1	ledc_ls_sig_out5	
85	rmt_sig_in2	ledc_ls_sig_out6	
86	rmt_sig_in3	ledc_ls_sig_out7	
87	rmt_sig_in4	rmt_sig_out0	
88	rmt_sig_in5	rmt_sig_out1	
89	rmt_sig_in6	rmt_sig_out2	
90	rmt_sig_in7	rmt_sig_out3	
91		rmt_sig_out4	
92		rmt_sig_out5	
93		rmt_sig_out6	

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
94		rmt_sig_out7	
95	I2CEXT1_SCL_in	I2CEXT1_SCL_out	
96	I2CEXT1_SDA_in	I2CEXT1_SDA_out	
97	host_card_detect_n_1	host_ccmd_od_pullup_en_n	
98	host_card_detect_n_2	host_rst_n_1	
99	host_card_write_prt_1	host_rst_n_2	
100	host_card_write_prt_2	gpio_sd0_out	
101	host_card_int_n_1	gpio_sd1_out	
102	host_card_int_n_2	gpio_sd2_out	
103	pwm1_sync0_in	gpio_sd3_out	
104	pwm1_sync1_in	gpio_sd4_out	
105	pwm1_sync2_in	gpio_sd5_out	
106	pwm1_f0_in	gpio_sd6_out	
107	pwm1_f1_in	gpio_sd7_out	
108	pwm1_f2_in	pwm1_out0a	
109	pwm0_cap0_in	pwm1_out0b	
110	pwm0_cap1_in	pwm1_out1a	
111	pwm0_cap2_in	pwm1_out1b	
112	pwm1_cap0_in	pwm1_out2a	
113	pwm1_cap1_in	pwm1_out2b	
114	pwm1_cap2_in	pwm2_out1h	
115	pwm2_flta	pwm2_out1l	
116	pwm2_fltb	pwm2_out2h	
117	pwm2_cap1_in	pwm2_out2l	
118	pwm2_cap2_in	pwm2_out3h	
119	pwm2_cap3_in	pwm2_out3l	
120	pwm3_flta	pwm2_out4h	
121	pwm3_fltb	pwm2_out4l	
122	pwm3_cap1_in		
123	pwm3_cap2_in		
124	pwm3_cap3_in		
140	I2S0I_DATA_in0	I2S0O_DATA_out0	
141	I2S0I_DATA_in1	I2S0O_DATA_out1	
142	I2S0I_DATA_in2	I2S0O_DATA_out2	
143	I2S0I_DATA_in3	I2S0O_DATA_out3	
144	I2S0I_DATA_in4	I2S0O_DATA_out4	
145	I2S0I_DATA_in5	I2S0O_DATA_out5	
146	I2S0I_DATA_in6	I2S0O_DATA_out6	
147	I2S0I_DATA_in7	I2S0O_DATA_out7	
148	I2S0I_DATA_in8	I2S0O_DATA_out8	
149	I2S0I_DATA_in9	I2S0O_DATA_out9	
150	I2S0I_DATA_in10	I2S0O_DATA_out10	
151	I2S0I_DATA_in11	I2S0O_DATA_out11	
152	I2S0I_DATA_in12	I2S0O_DATA_out12	

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
153	I2S0I_DATA_in13	I2S0O_DATA_out13	
154	I2S0I_DATA_in14	I2S0O_DATA_out14	
155	I2S0I_DATA_in15	I2S0O_DATA_out15	
156		I2S0O_DATA_out16	
157		I2S0O_DATA_out17	
158		I2S0O_DATA_out18	
159		I2S0O_DATA_out19	
160		I2S0O_DATA_out20	
161		I2S0O_DATA_out21	
162		I2S0O_DATA_out22	
163		I2S0O_DATA_out23	
164	I2S1I_BCK_in	I2S1I_BCK_out	
165	I2S1I_WS_in	I2S1I_WS_out	
166	I2S1I_DATA_in0	I2S1O_DATA_out0	
167	I2S1I_DATA_in1	I2S1O_DATA_out1	
168	I2S1I_DATA_in2	I2S1O_DATA_out2	
169	I2S1I_DATA_in3	I2S1O_DATA_out3	
170	I2S1I_DATA_in4	I2S1O_DATA_out4	
171	I2S1I_DATA_in5	I2S1O_DATA_out5	
172	I2S1I_DATA_in6	I2S1O_DATA_out6	
173	I2S1I_DATA_in7	I2S1O_DATA_out7	
174	I2S1I_DATA_in8	I2S1O_DATA_out8	
175	I2S1I_DATA_in9	I2S1O_DATA_out9	
176	I2S1I_DATA_in10	I2S1O_DATA_out10	
177	I2S1I_DATA_in11	I2S1O_DATA_out11	
178	I2S1I_DATA_in12	I2S1O_DATA_out12	
179	I2S1I_DATA_in13	I2S1O_DATA_out13	
180	I2S1I_DATA_in14	I2S1O_DATA_out14	
181	I2S1I_DATA_in15	I2S1O_DATA_out15	
182		I2S1O_DATA_out16	
183		I2S1O_DATA_out17	
184		I2S1O_DATA_out18	
185		I2S1O_DATA_out19	
186		I2S1O_DATA_out20	
187		I2S1O_DATA_out21	
188		I2S1O_DATA_out22	
189		I2S1O_DATA_out23	
190	I2S0I_H_SYNC	pwm3_out1h	
191	I2S0I_V_SYNC	pwm3_out1l	
192	I2S0I_H_ENABLE	pwm3_out2h	
193	I2S1I_H_SYNC	pwm3_out2l	
194	I2S1I_V_SYNC	pwm3_out3h	
195	I2S1I_H_ENABLE	pwm3_out3l	
196		pwm3_out4h	

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
197		pwm3_out4l	
198	U2RXD_in	U2TXD_out	YES
199	U2CTS_in	U2RTS_out	YES
200	emac_mdc_i	emac_mdc_o	
201	emac_mdi_i	emac_mdo_o	
202	emac_crs_i	emac_crs_o	
203	emac_col_i	emac_col_o	
204	pcmfsync_in	bt_audio0_irq	
205	pcmclk_in	bt_audio1_irq	
206	pcmdin	bt_audio2_irq	
207		ble_audio0_irq	
208		ble_audio1_irq	
209		ble_audio2_irq	
210		pcmfsync_out	
211		pcmclk_out	
212		pcmdout	
213		ble_audio_sync0_p	
214		ble_audio_sync1_p	
215		ble_audio_sync2_p	
224		sig_in_func224	
225		sig_in_func225	
226		sig_in_func226	
227		sig_in_func227	
228		sig_in_func228	

Direct I/O in IO_MUX "YES" means that this signal is also available directly via IO_MUX. To apply the GPIO Matrix to these signals, their corresponding SIG_IN_SEL register must be cleared.

4.10 IO_MUX Pad List

Table 17 shows the IO_MUX functions for each I/O pad:

Table 17: IO_MUX Pad Summary

GPIO	Pad Name	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	Reset	Notes
0	GPIO0	GPIO0	CLK_OUT1	GPIO0	-	-	EMAC_TX_CLK	3	R
1	U0TXD	U0TXD	CLK_OUT3	GPIO1	-	-	EMAC_RXD2	3	-
2	GPIO2	GPIO2	HSPIWP	GPIO2	HS2_DATA0	SD_DATA0	-	2	R
3	U0RXD	U0RXD	CLK_OUT2	GPIO3	-	-	-	3	-
4	GPIO4	GPIO4	HSPIHD	GPIO4	HS2_DATA1	SD_DATA1	EMAC_TX_ER	2	R
5	GPIO5	GPIO5	VSPICS0	GPIO5	HS1_DATA6	-	EMAC_RX_CLK	3	-
6	SD_CLK	SD_CLK	SPICLK	GPIO6	HS1_CLK	U1CTS	-	3	-
7	SD_DATA_0	SD_DATA0	SPIQ	GPIO7	HS1_DATA0	U2RTS	-	3	-
8	SD_DATA_1	SD_DATA1	SPID	GPIO8	HS1_DATA1	U2CTS	-	3	-
9	SD_DATA_2	SD_DATA2	SPIHD	GPIO9	HS1_DATA2	U1RXD	-	3	-
10	SD_DATA_3	SD_DATA3	SPIWP	GPIO10	HS1_DATA3	U1TXD	-	3	-
11	SD_CMD	SD_CMD	SPICS0	GPIO11	HS1_CMD	U1RTS	-	3	-
12	MTDI	MTDI	HSPIQ	GPIO12	HS2_DATA2	SD_DATA2	EMAC_TXD3	2	R

GPIO	Pad Name	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	Reset	Notes
13	MTCK	MTCK	HSPID	GPIO13	HS2_DATA3	SD_DATA3	EMAC_RX_ER	1	R
14	MTMS	MTMS	HSPICLK	GPIO14	HS2_CLK	SD_CLK	EMAC_TXD2	1	R
15	MTDO	MTDO	HSPICS0	GPIO15	HS2_CMD	SD_CMD	EMAC_RXD3	3	R
16	GPIO16	GPIO16	-	GPIO16	HS1_DATA4	U2RXD	EMAC_CLK_OUT	1	-
17	GPIO17	GPIO17	-	GPIO17	HS1_DATA5	U2TXD	EMAC_CLK_180	1	-
18	GPIO18	GPIO18	VSPICLK	GPIO18	HS1_DATA7	-	-	1	-
19	GPIO19	GPIO19	VSPIQ	GPIO19	U0CTS	-	EMAC_TXD0	1	-
20	GPIO20	GPIO20	-	GPIO20	-	-	-	1	-
21	GPIO21	GPIO21	VSPIHD	GPIO21	-	-	EMAC_TX_EN	1	-
22	GPIO22	GPIO22	VSPIWP	GPIO22	U0RTS	-	EMAC_TXD1	1	-
23	GPIO23	GPIO23	VSPID	GPIO23	HS1_STROBE	-	-	1	-
25	GPIO25	GPIO25	-	GPIO25	-	-	EMAC_RXD0	0	R
26	GPIO26	GPIO26	-	GPIO26	-	-	EMAC_RXD1	0	R
27	GPIO27	GPIO27	-	GPIO27	-	-	EMAC_RX_DV	1	R
32	32K_XP	GPIO32	-	GPIO32	-	-	-	0	R
33	32K_XN	GPIO33	-	GPIO33	-	-	-	0	R
34	VDET_1	GPIO34	-	GPIO34	-	-	-	0	R, I
35	VDET_2	GPIO35	-	GPIO35	-	-	-	0	R, I
36	SENSOR_VP	GPIO36	-	GPIO36	-	-	-	0	R, I
37	SENSOR_CAPP	GPIO37	-	GPIO37	-	-	-	0	R, I
38	SENSOR_CAPN	GPIO38	-	GPIO38	-	-	-	0	R, I
39	SENSOR_VN	GPIO39	-	GPIO39	-	-	-	0	R, I

Reset Configurations

”Reset” column shows each pad’s default configurations after reset:

- **0** - IE=0 (input disabled).
- **1** - IE=1 (input enabled).
- **2** - IE=1, WPD=1 (input enabled, pulldown resistor).
- **3** - IE=1, WPU=1 (input enabled, pullup resistor).

Notes

- **R** - Pad has RTC/analog functions via RTC_MUX.
- **I** - Pad can only be configured as input GPIO.

Refer to ESP32 Pin List datasheet for more details and a complete table of pin functions.

4.11 RTC_MUX Pin List

Table 18 shows the RTC pins and how they correspond to GPIO pads:

Table 18: RTC_MUX Pin Summary

RTC GPIO Num	GPIO Num	Pad Name	Analog Function		
			1	2	3
0	36	SENSOR_VP	ADC_H	ADC1_CH0	-
1	37	SENSOR_CAPP	ADC_H	ADC1_CH1	-
2	38	SENSOR_CAPN	ADC_H	ADC1_CH2	-
3	39	SENSOR_VN	ADC_H	ADC1_CH3	-
4	34	VDET_1	-	ADC1_CH6	-

RTC GPIO Num	GPIO Num	Pad Name	Analog Function		
			1	2	3
5	35	VDET_2	-	ADC1_CH7	-
6	25	GPIO25	DAC_1	ADC2_CH8	-
7	26	GPIO26	DAC_2	ADC2_CH9	-
8	33	32K_XN	XTAL_32K_N	ADC1_CH5	TOUCH8
9	32	32K_XP	XTAL_32K_P	ADC1_CH4	TOUCH9
10	4	GPIO4	-	ADC2_CH0	TOUCH0
11	0	GPIO0	-	ADC2_CH1	TOUCH1
12	2	GPIO2	-	ADC2_CH2	TOUCH2
13	15	MTDO	-	ADC2_CH3	TOUCH3
14	13	MTCK	-	ADC2_CH4	TOUCH4
15	12	MTDI	-	ADC2_CH5	TOUCH5
16	14	MTMS	-	ADC2_CH6	TOUCH6
17	27	GPIO27	-	ADC2_CH7	TOUCH7

4.12 Register Summary

Name	Description	Address	Access
GPIO_OUT_REG	GPIO 0-31 output register_REG	0x3FF44004	R/W
GPIO_OUT_W1TS_REG	GPIO 0-31 output register_W1TS_REG	0x3FF44008	RO
GPIO_OUT_W1TC_REG	GPIO 0-31 output register_W1TC_REG	0x3FF4400C	RO
GPIO_OUT1_REG	GPIO 32-39 output register_REG	0x3FF44010	R/W
GPIO_OUT1_W1TS_REG	GPIO 32-39 output bit set register_REG	0x3FF44014	RO
GPIO_OUT1_W1TC_REG	GPIO 32-39 output bit clear register_REG	0x3FF44018	RO
GPIO_ENABLE_REG	GPIO 0-31 output enable register_REG	0x3FF44020	R/W
GPIO_ENABLE_W1TS_REG	GPIO 0-31 output enable register_W1TS_REG	0x3FF44024	RO
GPIO_ENABLE_W1TC_REG	GPIO 0-31 output enable register_W1TC_REG	0x3FF44028	RO
GPIO_ENABLE1_REG	GPIO 32-39 output enable register_REG	0x3FF4402C	R/W
GPIO_ENABLE1_W1TS_REG	GPIO 32-39 output enable bit set register_REG	0x3FF44030	RO
GPIO_ENABLE1_W1TC_REG	GPIO 32-39 output enable bit clear register_REG	0x3FF44034	RO
GPIO_STRAP_REG	Bootstrap pin value register_REG	0x3FF44038	RO
GPIO_IN_REG	GPIO 0-31 input register_REG	0x3FF4403C	RO
GPIO_IN1_REG	GPIO 32-39 input register_REG	0x3FF44040	RO
GPIO_STATUS_REG	GPIO 0-31 interrupt status register_REG	0x3FF44044	R/W
GPIO_STATUS_W1TS_REG	GPIO 0-31 interrupt status register_W1TS_REG	0x3FF44048	RO
GPIO_STATUS_W1TC_REG	GPIO 0-31 interrupt status register_W1TC_REG	0x3FF4404C	RO
GPIO_STATUS1_REG	GPIO 32-39 interrupt status register1_REG	0x3FF44050	R/W
GPIO_STATUS1_W1TS_REG	GPIO 32-39 interrupt status bit set register_REG	0x3FF44054	RO
GPIO_STATUS1_W1TC_REG	GPIO 32-39 interrupt status bit clear register_REG	0x3FF44058	RO
GPIO_ACPU_INT_REG	GPIO 0-31 APP_CPU interrupt status_REG	0x3FF44060	RO
GPIO_ACPU_NMI_INT_REG	GPIO 0-31 APP_CPU non-maskable interrupt status_REG	0x3FF44064	RO
GPIO_PCPU_INT_REG	GPIO 0-31 PRO_CPU interrupt status_REG	0x3FF44068	RO

Name	Description	Address	Access
GPIO_PCPU_NMI_INT_REG	GPIO 0-31 PRO_CPU non-maskable interrupt status_REG	0x3FF4406C	RO
GPIO_ACPU_INT1_REG	GPIO 32-39 APP_CPU interrupt status_REG	0x3FF44074	RO
GPIO_ACPU_NMI_INT1_REG	GPIO 32-39 APP_CPU non-maskable interrupt status_REG	0x3FF44078	RO
GPIO_PCPU_INT1_REG	GPIO 32-39 PRO_CPU interrupt status_REG	0x3FF4407C	RO
GPIO_PCPU_NMI_INT1_REG	GPIO 32-39 PRO_CPU non-maskable interrupt status_REG	0x3FF44080	RO
GPIO_PIN0_REG	Configuration for GPIO pin 0_REG	0x3FF44088	R/W
GPIO_PIN1_REG	Configuration for GPIO pin 1_REG	0x3FF4408C	R/W
GPIO_PIN2_REG	Configuration for GPIO pin 2_REG	0x3FF44090	R/W
...	...		
GPIO_PIN38_REG	Configuration for GPIO pin 38_REG	0x3FF44120	R/W
GPIO_PIN39_REG	Configuration for GPIO pin 39_REG	0x3FF44124	R/W
GPIO_FUNC0_IN_SEL_CFG_REG	Peripheral function 0 input selection register_REG	0x3FF44130	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Peripheral function 1 input selection register_REG	0x3FF44134	R/W
...	...		
GPIO_FUNC254_IN_SEL_CFG_REG	Peripheral function 254 input selection register_REG	0x3FF44528	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	Peripheral function 255 input selection register_REG	0x3FF4452C	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 0_REG	0x3FF44530	R/W
GPIO_FUNC1_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 1_REG	0x3FF44534	R/W
...	...		
GPIO_FUNC38_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 38_REG	0x3FF445C8	R/W
GPIO_FUNC39_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 39_REG	0x3FF445CC	R/W

Name	Description	Address	Access
IO_MUX_GPIO36_REG	Configuration register for pad GPIO36	0x3FF49004	R/W
IO_MUX_GPIO37_REG	Configuration register for pad GPIO37	0x3FF49008	R/W
IO_MUX_GPIO38_REG	Configuration register for pad GPIO38	0x3FF4900C	R/W
IO_MUX_GPIO39_REG	Configuration register for pad GPIO39	0x3FF49010	R/W
IO_MUX_GPIO34_REG	Configuration register for pad GPIO34	0x3FF49014	R/W
IO_MUX_GPIO35_REG	Configuration register for pad GPIO35	0x3FF49018	R/W
IO_MUX_GPIO32_REG	Configuration register for pad GPIO32	0x3FF4901C	R/W
IO_MUX_GPIO33_REG	Configuration register for pad GPIO33	0x3FF49020	R/W
IO_MUX_GPIO25_REG	Configuration register for pad GPIO25	0x3FF49024	R/W
IO_MUX_GPIO26_REG	Configuration register for pad GPIO26	0x3FF49028	R/W
IO_MUX_GPIO27_REG	Configuration register for pad GPIO27	0x3FF4902C	R/W
IO_MUX_MTMS_REG	Configuration register for pad MTMS	0x3FF49030	R/W
IO_MUX_MTDI_REG	Configuration register for pad MTDI	0x3FF49034	R/W
IO_MUX_MTCK_REG	Configuration register for pad MTCK	0x3FF49038	R/W
IO_MUX_MTDO_REG	Configuration register for pad MTDO	0x3FF4903C	R/W
IO_MUX_GPIO2_REG	Configuration register for pad GPIO2	0x3FF49040	R/W

Name	Description	Address	Access
IO_MUX_GPIO0_REG	Configuration register for pad GPIO0	0x3FF49044	R/W
IO_MUX_GPIO4_REG	Configuration register for pad GPIO4	0x3FF49048	R/W
IO_MUX_GPIO16_REG	Configuration register for pad GPIO16	0x3FF4904C	R/W
IO_MUX_GPIO17_REG	Configuration register for pad GPIO17	0x3FF49050	R/W
IO_MUX_SD_DATA2_REG	Configuration register for pad SD_DATA2	0x3FF49054	R/W
IO_MUX_SD_DATA3_REG	Configuration register for pad SD_DATA3	0x3FF49058	R/W
IO_MUX_SD_CMD_REG	Configuration register for pad SD_CMD	0x3FF4905C	R/W
IO_MUX_SD_CLK_REG	Configuration register for pad SD_CLK	0x3FF49060	R/W
IO_MUX_SD_DATA0_REG	Configuration register for pad SD_DATA0	0x3FF49064	R/W
IO_MUX_SD_DATA1_REG	Configuration register for pad SD_DATA1	0x3FF49068	R/W
IO_MUX_GPIO5_REG	Configuration register for pad GPIO5	0x3FF4906C	R/W
IO_MUX_GPIO18_REG	Configuration register for pad GPIO18	0x3FF49070	R/W
IO_MUX_GPIO19_REG	Configuration register for pad GPIO19	0x3FF49074	R/W
IO_MUX_GPIO20_REG	Configuration register for pad GPIO20	0x3FF49078	R/W
IO_MUX_GPIO21_REG	Configuration register for pad GPIO21	0x3FF4907C	R/W
IO_MUX_GPIO22_REG	Configuration register for pad GPIO22	0x3FF49080	R/W
IO_MUX_U0RXD_REG	Configuration register for pad U0RXD	0x3FF49084	R/W
IO_MUX_U0TXD_REG	Configuration register for pad U0TXD	0x3FF49088	R/W
IO_MUX_GPIO23_REG	Configuration register for pad GPIO23	0x3FF4908C	R/W
IO_MUX_GPIO24_REG	Configuration register for pad GPIO24	0x3FF49090	R/W

Name	Description	Address	Access
GPIO configuration / data registers			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO output register_REG	0x3FF48000	R/W
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO output bit set register_REG	0x3FF48004	WO
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO output bit clear register_REG	0x3FF48008	WO
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO output enable register_REG	0x3FF4800C	R/W
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable bit setregister_REG	0x3FF48010	WO
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable bit clear register_REG	0x3FF48014	WO
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO interrupt status register_REG	0x3FF48018	WO
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status bit set register_REG	0x3FF4801C	WO
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status bit clear register_REG	0x3FF48020	WO
RTCIO_RTC_GPIO_IN_REG	RTC GPIO input register_REG	0x3FF48024	RO
RTCIO_RTC_GPIO_PIN0_REG	RTC configuration for pin 0_REG	0x3FF48028	R/W
RTCIO_RTC_GPIO_PIN1_REG	RTC configuration for pin 1_REG	0x3FF4802C	R/W
RTCIO_RTC_GPIO_PIN2_REG	RTC configuration for pin 2_REG	0x3FF48030	R/W
RTCIO_RTC_GPIO_PIN3_REG	RTC configuration for pin 3_REG	0x3FF48034	R/W
RTCIO_RTC_GPIO_PIN4_REG	RTC configuration for pin 4_REG	0x3FF48038	R/W
RTCIO_RTC_GPIO_PIN5_REG	RTC configuration for pin 5_REG	0x3FF4803C	R/W
RTCIO_RTC_GPIO_PIN6_REG	RTC configuration for pin 6_REG	0x3FF48040	R/W
RTCIO_RTC_GPIO_PIN7_REG	RTC configuration for pin 7_REG	0x3FF48044	R/W
RTCIO_RTC_GPIO_PIN8_REG	RTC configuration for pin 8_REG	0x3FF48048	R/W
RTCIO_RTC_GPIO_PIN9_REG	RTC configuration for pin 9_REG	0x3FF4804C	R/W

Name	Description	Address	Access
RTCIO_RTC_GPIO_PIN10_REG	RTC configuration for pin 10_REG	0x3FF48050	R/W
RTCIO_RTC_GPIO_PIN11_REG	RTC configuration for pin 11_REG	0x3FF48054	R/W
RTCIO_RTC_GPIO_PIN12_REG	RTC configuration for pin 12_REG	0x3FF48058	R/W
RTCIO_RTC_GPIO_PIN13_REG	RTC configuration for pin 13_REG	0x3FF4805C	R/W
RTCIO_RTC_GPIO_PIN14_REG	RTC configuration for pin 14_REG	0x3FF48060	R/W
RTCIO_RTC_GPIO_PIN15_REG	RTC configuration for pin 15_REG	0x3FF48064	R/W
RTCIO_RTC_GPIO_PIN16_REG	RTC configuration for pin 16_REG	0x3FF48068	R/W
RTCIO_RTC_GPIO_PIN17_REG	RTC configuration for pin 17_REG	0x3FF4806C	R/W
RTCIO_DIG_PAD_HOLD_REG	RTC GPIO hold register_REG	0x3FF48074	R/W
GPIO RTC function configuration registers			
RTCIO_HALL_SENS_REG	Hall sensor configuration_REG	0x3FF48078	R/W
RTCIO_SENSOR_PADS_REG	Sensor pads configuration register_REG	0x3FF4807C	R/W
RTCIO_ADC_PAD_REG	ADC configuration register_REG	0x3FF48080	R/W
RTCIO_PAD_DAC1_REG	DAC1 configuration register_REG	0x3FF48084	R/W
RTCIO_PAD_DAC2_REG	DAC2 configuration register_REG	0x3FF48088	R/W
RTCIO_XTAL_32K_PAD_REG	32KHz crystal pads configuration register_REG	0x3FF4808C	R/W
RTCIO_TOUCH_CFG_REG	Touch sensor configuration register_REG	0x3FF48090	R/W
RTCIO_TOUCH_PAD0_REG	Touch pad configuration register_REG	0x3FF48094	R/W
...	...		
RTCIO_TOUCH_PAD9_REG	Touch pad configuration register_REG	0x3FF480B8	R/W
RTCIO_EXT_WAKEUP0_REG	External wake up configuration register_REG	0x3FF480BC	R/W
RTCIO_XTL_EXT_CTR_REG	Crystal power down enable gpio source_REG	0x3FF480C0	R/W
RTCIO_SAR_I2C_IO_REG	RTC I2C pad selection_REG	0x3FF480C4	R/W

4.13 Registers

Register 4.1: GPIO_OUT_REG (0x0004)

31																															0	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reset

GPIO_OUT_REG GPIO0-31 output value. (R/W)

Register 4.2: GPIO_OUT_W1TS_REG (0x0008)

31																															0	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reset

GPIO_OUT_W1TS_REG GPIO0-31 output set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_DATA will be set. (RO)

Register 4.3: GPIO_OUT_W1TC_REG (0x000c)

31																															0	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reset

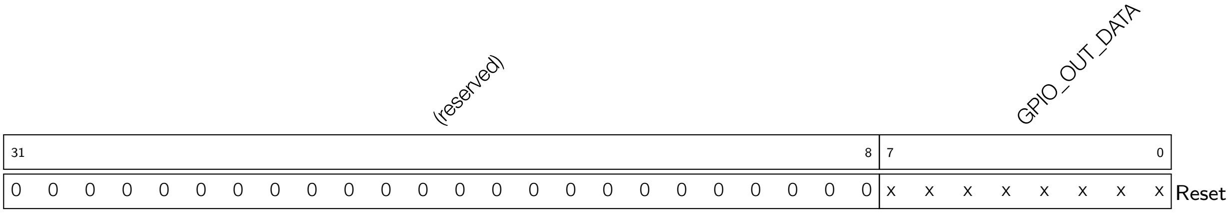
GPIO_OUT_W1TC_REG GPIO0-31 output clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_DATA will be cleared. (RO)

Register 4.4: GPIO_OUT1_REG (0x0010)

(reserved)																								GPIO_OUT_DATA								
31																							8	7							0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	Reset

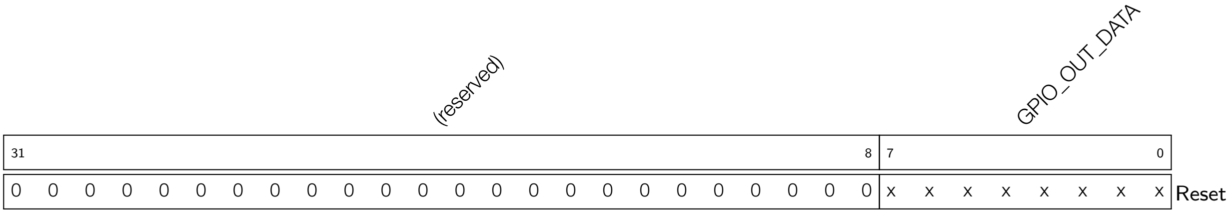
GPIO_OUT_DATA GPIO32-39 output value. (R/W)

Register 4.5: GPIO_OUT1_W1TS_REG (0x0014)



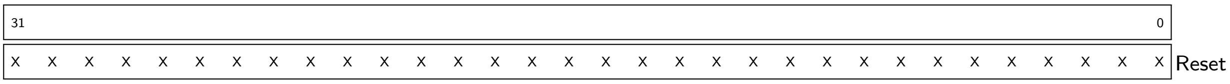
GPIO_OUT_DATA GPIO32-39 output value set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT1_DATA will be set. (RO)

Register 4.6: GPIO_OUT1_W1TC_REG (0x0018)



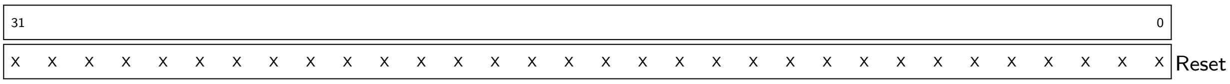
GPIO_OUT_DATA GPIO32-39 output value clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT1_DATA will be cleared. (RO)

Register 4.7: GPIO_ENABLE_REG (0x0020)



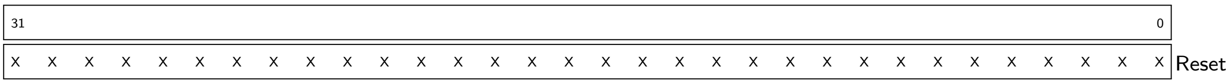
GPIO_ENABLE_REG GPIO0-31 output enable. (R/W)

Register 4.8: GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS_REG GPIO0-31 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE will be set. (RO)

Register 4.9: GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC_REG GPIO0-31 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE will be cleared. (RO)

Register 4.10: GPIO_ENABLE1_REG (0x002c)

(reserved)	GPIO_ENABLE_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_ENABLE_DATA GPIO32-39 output enable. (R/W)

Register 4.11: GPIO_ENABLE1_W1TS_REG (0x0030)

(reserved)	GPIO_ENABLE_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_ENABLE_DATA GPIO32-39 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE1 will be set. (RO)

Register 4.12: GPIO_ENABLE1_W1TC_REG (0x0034)

(reserved)	GPIO_ENABLE_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_ENABLE_DATA GPIO32-39 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE1 will be cleared. (RO)

Register 4.13: GPIO_STRAP_REG (0x0038)

(reserved)	GPIO_STRAPPING	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x x x x x x x x x	Reset

GPIO_STRAPPING GPIO strapping results: boot_sel_chip[5:0]: MTDI, GPIO0, GPIO2, GPIO4, MTDO, GPIO5.

Register 4.14: GPIO_IN_REG (0x003c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_IN_REG GPIO0-31 input value. Each bit represents a pad input value, 1 for high level and 0 for low level. (RO)

Register 4.15: GPIO_IN1_REG (0x0040)

(reserved)	GPIO_IN_DATA_NEXT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_IN_DATA_NEXT GPIO32-39 input value. Each bit represents a pad input value. (RO)

Register 4.16: GPIO_STATUS_REG (0x0044)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_STATUS_REG GPIO0-31 interrupt status register. Each bit can be either of the two interrupt sources for the two CPUs. The enable bits in GPIO_STATUS_INTERRUPT, corresponding to the 0-4 bits in GPIO_PIN*n*_REG should be set to 1. (R/W)

Register 4.17: GPIO_STATUS_W1TS_REG (0x0048)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

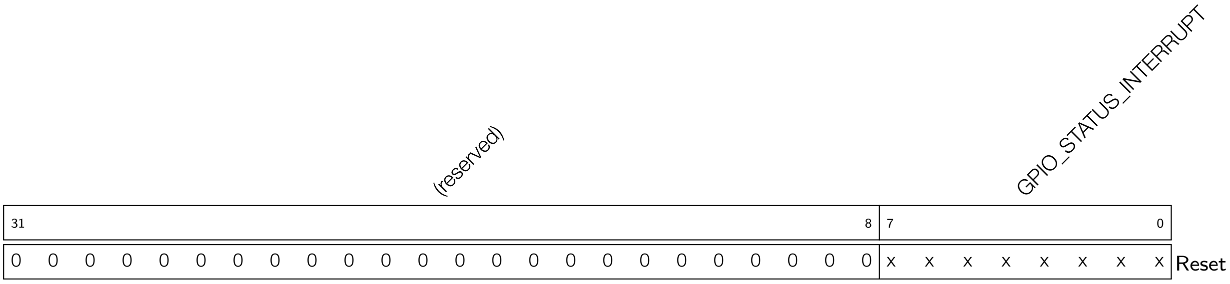
GPIO_STATUS_W1TS_REG GPIO0-31 interrupt status set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set. (RO)

Register 4.18: GPIO_STATUS_W1TC_REG (0x004c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

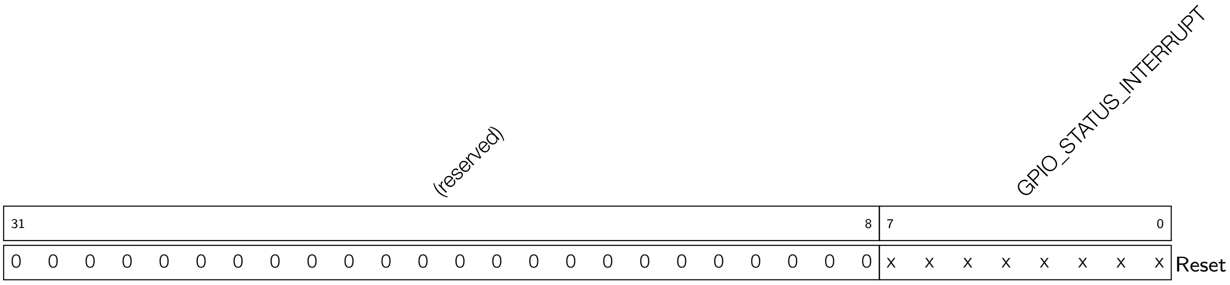
GPIO_STATUS_W1TC_REG GPIO0-31 interrupt status clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. (RO)

Register 4.19: GPIO_STATUS1_REG (0x0050)



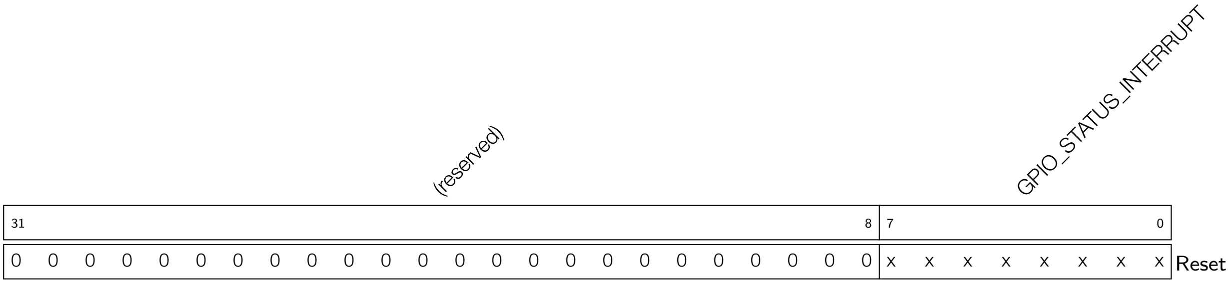
GPIO_STATUS_INTERRUPT GPIO32-39 interrupt status. (R/W)

Register 4.20: GPIO_STATUS1_W1TS_REG (0x0054)



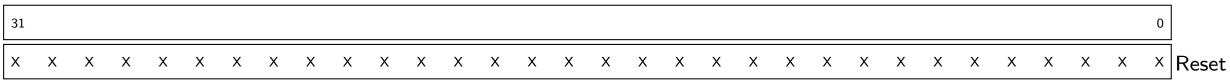
GPIO_STATUS_INTERRUPT GPIO32-39 interrupt status set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT1 will be set. (RO)

Register 4.21: GPIO_STATUS1_W1TC_REG (0x0058)



GPIO_STATUS_INTERRUPT GPIO32-39 interrupt status clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT1 will be cleared. (RO)

Register 4.22: GPIO_ACPU_INT_REG (0x0060)



GPIO_ACPU_INT_REG GPIO0-31 APP CPU interrupt status. (RO)

Register 4.23: GPIO_ACPU_NMI_INT_REG (0x0064)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ACPU_NMI_INT_REG GPIO0-31 APP CPU non-maskable interrupt status. (RO)

Register 4.24: GPIO_PCPU_INT_REG (0x0068)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_PCPU_INT_REG GPIO0-31 PRO CPU interrupt status. (RO)

Register 4.25: GPIO_PCPU_NMI_INT_REG (0x006c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_PCPU_NMI_INT_REG GPIO0-31 PRO CPU non-maskable interrupt status. (RO)

Register 4.26: GPIO_ACPU_INT1_REG (0x0074)

(reserved)	GPIO_APPCPU_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

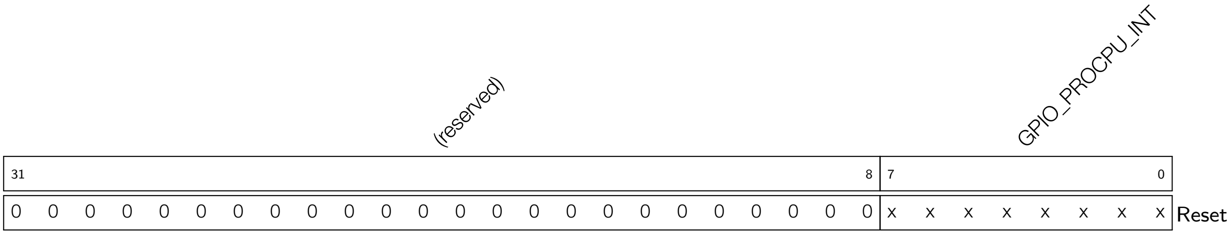
GPIO_APPCPU_INT GPIO32-39 APP CPU interrupt status. (RO)

Register 4.27: GPIO_ACPU_NMI_INT1_REG (0x0078)

(reserved)	GPIO_APPCPU_NMI_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

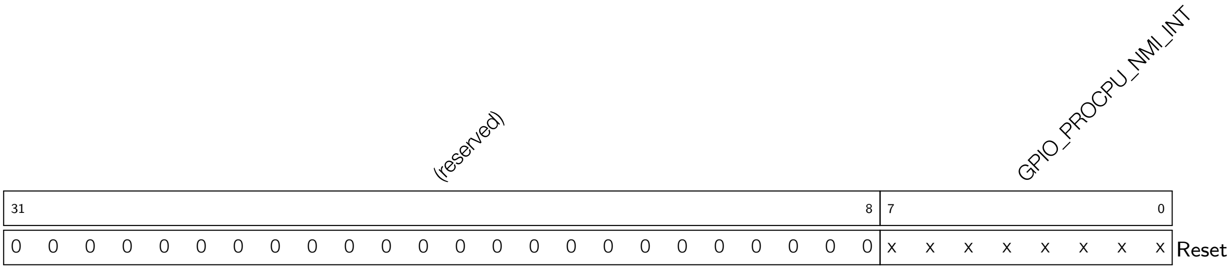
GPIO_APPCPU_NMI_INT GPIO32-39 APP CPU non-maskable interrupt status. (RO)

Register 4.28: GPIO_PCPU_INT1_REG (0x007c)



GPIO_PROCPU_INT GPIO32-39 PRO CPU interrupt status. (RO)

Register 4.29: GPIO_PCPU_NMI_INT1_REG (0x0080)



GPIO_PROCPU_NMI_INT GPIO32-39 PRO CPU non-maskable interrupt status. (RO)

Register 4.30: GPIO_PIN*n*_REG (*n*: 0-39) (0x88+0x4**n*)

	(reserved)	GPIO_PIN*n*_INT_ENA	(reserved)	GPIO_PIN*n*_WAKEUP_ENABLE	GPIO_PIN*n*_INT_TYPE	(reserved)	GPIO_PIN*n*_PAD_DRIVER	(reserved)
	31 18	17 13	12 11	10	9 7	6 3	2	3 2
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x	0 0	x	x x x	0 0 0 0	x	0 0

GPIO_PIN*n*_INT_ENA Interrupt enable bits for pin *n*: (R/W)

bit0: APP CPU interrupt enable;
bit1: APP CPU non-maskable interrupt enable;
bit3: PRO CPU interrupt enable;
bit4: PRO CPU non-maskable interrupt enable.

GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable will only wake up the CPU from Light-sleep. (R/W)

GPIO_PIN*n*_INT_TYPE Interrupt type selection: (R/W)

0: GPIO interrupt disable;
1: rising edge trigger;
2: falling edge trigger;
3: any edge trigger;
4: low level trigger;
5: high level trigger.

GPIO_PIN*n*_PAD_DRIVER 0: normal output; 1: open drain output. (R/W)

Register 4.31: GPIO_FUNC*m*_IN_SEL_CFG_REG (*m*: 0-255) (0x130+0x4**m*)

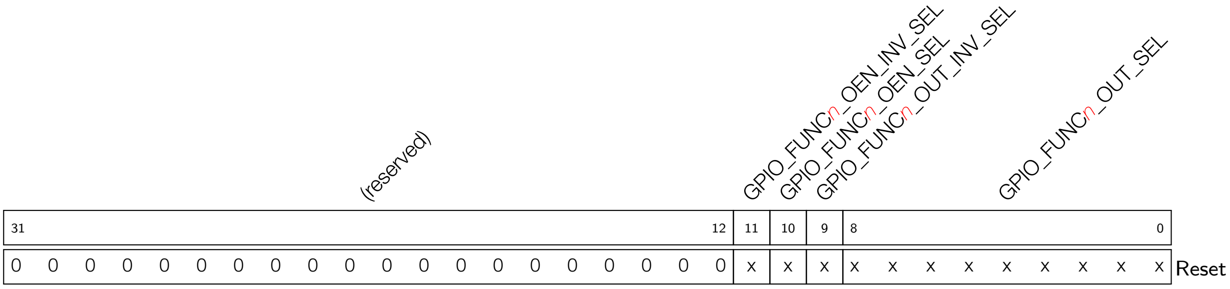
	(reserved)	GPIO_SIG*m*_IN_SEL	GPIO_FUNC*m*_IN_INV_SEL	GPIO_FUNC*m*_IN_SEL
	31 8	7	6	5 0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x	x	x x x x x x

GPIO_SIG*m*_IN_SEL Bypass the GPIO Matrix. 0: route through GPIO Matrix, 1: connect signal directly to peripheral configured in the IO_MUX. (R/W)

GPIO_FUNC*m*_IN_INV_SEL Invert the input value. 1: invert; 0: do not invert. (R/W)

GPIO_FUNC*m*_IN_SEL Selection control for peripheral input *m*. A value of 0-39 selects which of the 40 GPIO Matrix input pins this signal is connected to, or 0x38 for a constantly high input or 0x30 for a constantly low input. (R/W)

Register 4.32: GPIO_FUNCn_OUT_SEL_CFG_REG (n: 0-39) (0x530+0x4*n)



GPIO_FUNCn_OEN_INV_SEL 1: Invert the output enable signal; 0: do not invert the output enable signal. (R/W)

GPIO_FUNCn_OEN_SEL 1: Force the output enable signal to be sourced from bit n of GPIO_ENABLE_REG; 0: use output enable signal from peripheral. (R/W)

GPIO_FUNCn_OUT_INV_SEL 1: Invert the output value; 0: do not invert the output value. (R/W)

GPIO_FUNCn_OUT_SEL Selection control for GPIO output n. A value of s(0<=s<256) connects peripheral output s to GPIO output n. A value of 256 selects bit n of GPIO_DATA_REG and GPIO_ENABLE_REG as the output value and output enable. (R/W)

Register 4.33: IO_MUX_*x*_REG (*x*: GPIO0-GPIO39) (0x10+4**x*)

(reserved)	IO_*x*_MCU_SEL	IO_*x*_FUNC_DRV	IO_*x*_FUNC_IE	IO_*x*_FUNC_WPU	IO_*x*_FUNC_WPD	IO_*x*_MCU_DRV	IO_*x*_MCU_IE	IO_*x*_MCU_WPU	IO_*x*_MCU_WPD	IO_*x*_SLP_SEL	IO_*x*_MCU_OE	
31 15	14 12	11 10	9	8	7	6 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x2	0	0	0	0x0	0	0	0	0	0	Reset

IO_*x*_MCU_SEL Select the IO_MUX function for this signal. 0 selects Function 1, 1 selects Function 2, etc. (R/W)

IO_*x*_FUNC_DRV Select the drive strength of the pad. A higher value corresponds with a higher strength. (R/W)

IO_*x*_FUNC_IE Input enable of the pad. 1: input enabled; 0: input disabled. (R/W)

IO_*x*_FUNC_WPU Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

IO_*x*_FUNC_WPD Pull-down enable of the pad. 1: internal pull-down enabled, 0: internal pull-down disabled. (R/W)

IO_*x*_MCU_DRV Select the drive strength of the pad during sleep mode. A higher value corresponds with a higher strength. (R/W)

IO_*x*_MCU_IE Input enable of the pad during sleep mode. 1: input enabled; 0: input disabled. (R/W)

IO_*x*_MCU_WPU Pull-up enable of the pad during sleep mode. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

IO_*x*_MCU_WPD Pull-down enable of the pad during sleep mode. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

IO_*x*_SLP_SEL Sleep mode selection of this pad. Set to 1 to put the pad in sleep mode. (R/W)

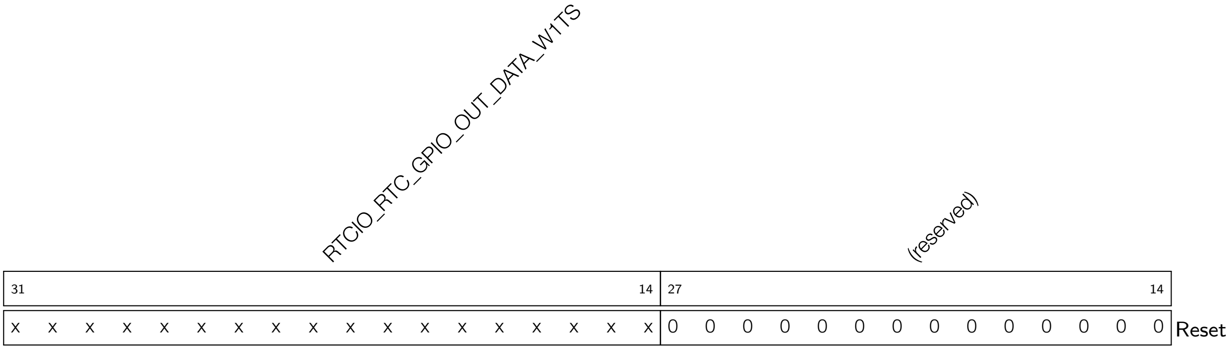
IO_*x*_MCU_OE Output enable of the pad in sleep mode. 1: enable output; 0: disable output. (R/W)

Register 4.34: RTCIO_RTC_GPIO_OUT_REG (0x0000)

RTCIO_RTC_GPIO_OUT_DATA	(reserved)	
31 14	27 14	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

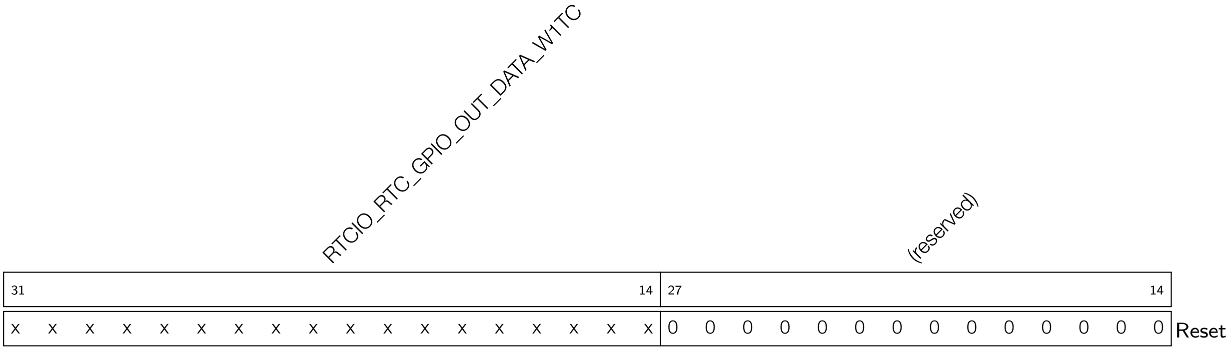
RTCIO_RTC_GPIO_OUT_DATA GPIO0-17 output register. Bit14 is GPIO[0], bit15 is GPIO[1], etc. (R/W)

Register 4.35: RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0001)



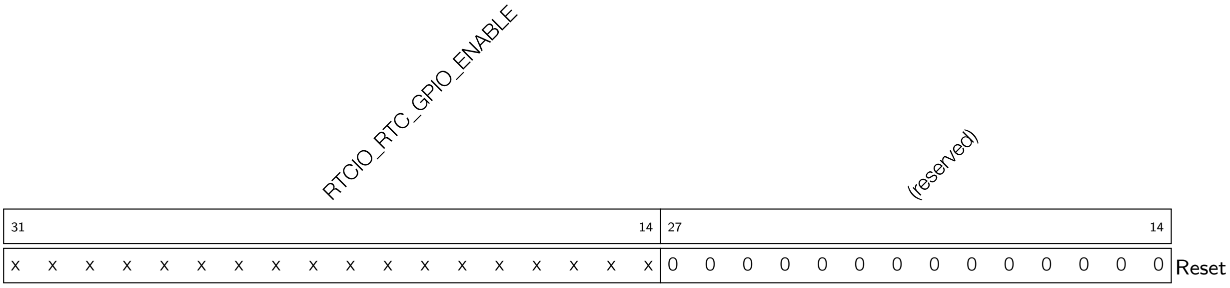
RTCIO_RTC_GPIO_OUT_DATA_W1TS GPIO0-17 output set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_OUT will be set. (WO)

Register 4.36: RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0002)



RTCIO_RTC_GPIO_OUT_DATA_W1TC GPIO0-17 output clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_OUT will be cleared. (WO)

Register 4.37: RTCIO_RTC_GPIO_ENABLE_REG (0x0003)



RTCIO_RTC_GPIO_ENABLE GPIO0-17 output enable. Bit14 is GPIO[0], bit15 is GPIO[1], etc. 1 means this GPIO pad is output. (R/W)

Register 4.38: RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0004)

RTCIO_RTC_GPIO_ENABLE_W1TS	(reserved)	
31 14	27 14	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_ENABLE_W1TS GPIO0-17 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE will be set. (WO)

Register 4.39: RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0005)

RTCIO_RTC_GPIO_ENABLE_W1TC	(reserved)	
31 14	27 14	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

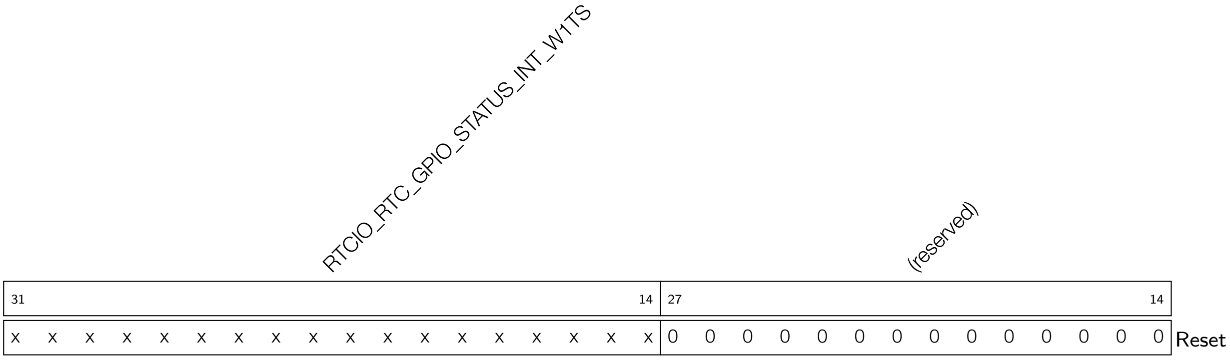
RTCIO_RTC_GPIO_ENABLE_W1TC GPIO0-17 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE will be cleared. (WO)

Register 4.40: RTCIO_RTC_GPIO_STATUS_REG (0x0006)

RTCIO_RTC_GPIO_STATUS_INT	(reserved)	
31 14	27 14	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

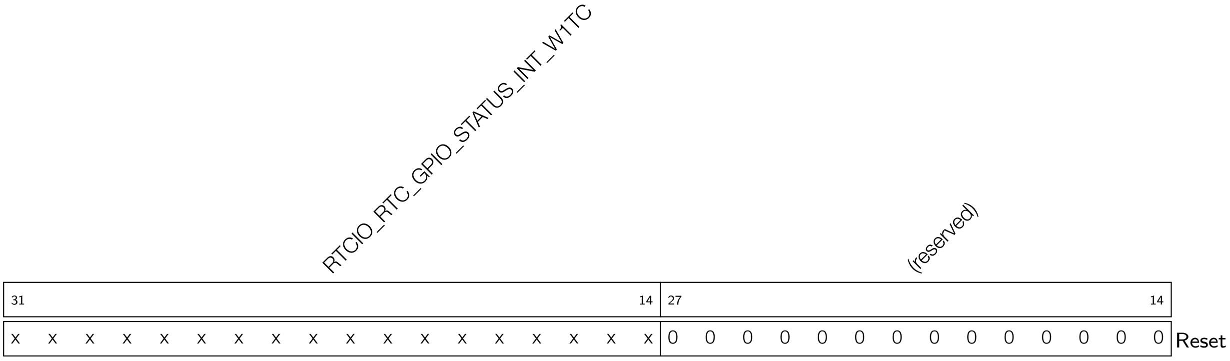
RTCIO_RTC_GPIO_STATUS_INT GPIO0-17 interrupt status. Bit14 is GPIO[0], bit15 is GPIO[1], etc. This register should be used together with RTCIO_RTC_GPIO_PIN*n*_INT_TYPE in RTCIO_RTC_GPIO_PIN*n*_REG. 1: corresponding interrupt; 0: no interrupt. (R/W)

Register 4.41: RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x0007)



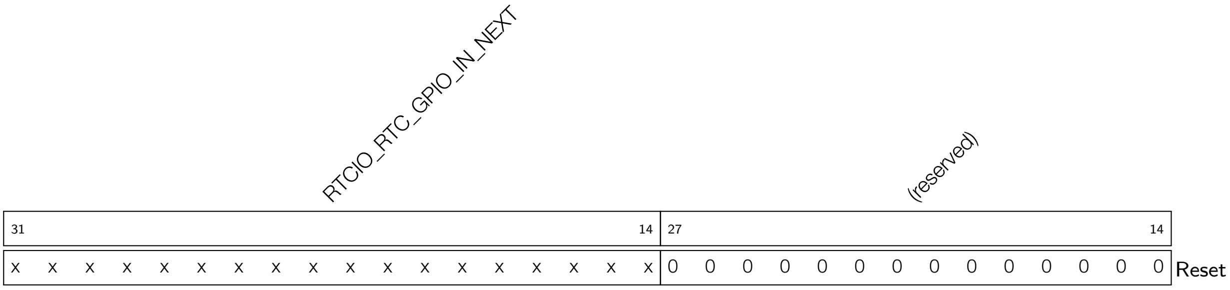
RTCIO_RTC_GPIO_STATUS_INT_W1TS GPIO0-17 interrupt set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_STATUS_INT will be set. (WO)

Register 4.42: RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0008)



RTCIO_RTC_GPIO_STATUS_INT_W1TC GPIO0-17 interrupt clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_STATUS_INT will be cleared. (WO)

Register 4.43: RTCIO_RTC_GPIO_IN_REG (0x0009)



RTCIO_RTC_GPIO_IN_NEXT GPIO0-17 input value. Bit14 is GPIO[0], bit15 is GPIO[1], etc. Each bit represents a pad input value, 1 for high level, and 0 for low level. (RO)

Register 4.44: RTCIO_RTC_GPIO_PIN*n*_REG (*n*: 0-17) (0xA+1***n*)

(reserved)	RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE	RTCIO_RTC_GPIO_PIN*n*_INT_TYPE	(reserved)	RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER	(reserved)	
31 11	10	9 7	6 3	2	3 2	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x	x x x	0 0 0 0	x	0 0	Reset

RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable. This will only wake up the ESP32 from Light-sleep. (R/W)

RTCIO_RTC_GPIO_PIN*n*_INT_TYPE GPIO interrupt type selection. (R/W)

0: GPIO interrupt disable;

1: rising edge trigger;

2: falling edge trigger;

3: any edge trigger;

4: low level trigger;

5: high level trigger.

RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER Pad driver selection. 0: normal output; 1: open drain. (R/W)

Register 4.45: RTCIO_DIG_PAD_HOLD_REG (0x001d)

31 0	
0	Reset

RTCIO_DIG_PAD_HOLD_REG Select which digital pads are on hold. While 0 allows normal operation, 1 puts the pad on hold. (R/W)

Register 4.46: RTCIO_HALL_SENS_REG (0x001e)

RTCIO_HALL_XPD_HALL	RTCIO_HALL_PHASE	(reserved)	
31	30	59 30	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_HALL_XPD_HALL Power on hall sensor and connect to VP and VN. (R/W)

RTCIO_HALL_PHASE Reverse the polarity of the hall sensor. (R/W)

Register 4.47: RTCIO_SENSOR_PADS_REG (0x001f)

RTCIO_SENSOR_SENSE1_HOLD	RTCIO_SENSOR_SENSE2_HOLD	RTCIO_SENSOR_SENSE3_HOLD	RTCIO_SENSOR_SENSE4_HOLD	RTCIO_SENSOR_SENSE1_MUX_SEL	RTCIO_SENSOR_SENSE2_MUX_SEL	RTCIO_SENSOR_SENSE3_MUX_SEL	RTCIO_SENSOR_SENSE4_MUX_SEL	RTCIO_SENSOR_SENSE1_FUN_SEL	RTCIO_SENSOR_SENSE1_SLP_SEL	RTCIO_SENSOR_SENSE1_SLP_IE	RTCIO_SENSOR_SENSE1_FUN_IE	RTCIO_SENSOR_SENSE2_FUN_SEL	RTCIO_SENSOR_SENSE2_SLP_SEL	RTCIO_SENSOR_SENSE2_SLP_IE	RTCIO_SENSOR_SENSE2_FUN_IE	RTCIO_SENSOR_SENSE3_FUN_SEL	RTCIO_SENSOR_SENSE3_SLP_SEL	RTCIO_SENSOR_SENSE3_SLP_IE	RTCIO_SENSOR_SENSE3_FUN_IE	RTCIO_SENSOR_SENSE4_FUN_SEL	RTCIO_SENSOR_SENSE4_SLP_SEL	RTCIO_SENSOR_SENSE4_SLP_IE	RTCIO_SENSOR_SENSE4_FUN_IE	(reserved)	
31	30	29	28	27	26	25	24	23 22	21	20	19	18 17	16	15	14	13 12	11	10	9	8 7	6	5	4	7 4	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0	Reset

RTCIO_SENSOR_SENSE*n*_HOLD Set to 1 to hold the output value on sense*n*; 0 is for normal operation. (R/W)

RTCIO_SENSOR_SENSE*n*_MUX_SEL 1: route sense*n* to the RTC block; 0: route sense*n* to the digital IO_MUX. (R/W)

RTCIO_SENSOR_SENSE*n*_FUN_SEL Select the RTC IO_MUX function for this pad. 0: select Function 0; 1: select Function 1. (R/W)

RTCIO_SENSOR_SENSE*n*_SLP_SEL Selection of sleep mode for the pad: set to 1 to put the pad in sleep mode. (R/W)

RTCIO_SENSOR_SENSE*n*_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_SENSOR_SENSE*n*_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

Register 4.48: RTCIO_ADC_PAD_REG (0x0020)

RTCIO_ADC_ADC1_HOLD	RTCIO_ADC_ADC2_HOLD	RTCIO_ADC_ADC1_MUX_SEL	RTCIO_ADC_ADC2_MUX_SEL	RTCIO_ADC_ADC1_FUN_SEL	RTCIO_ADC_ADC1_SLP_SEL	RTCIO_ADC_ADC1_SLP_IE	RTCIO_ADC_ADC1_FUN_IE	RTCIO_ADC_ADC2_FUN_SEL	RTCIO_ADC_ADC2_SLP_SEL	RTCIO_ADC_ADC2_SLP_IE	RTCIO_ADC_ADC2_FUN_IE	(reserved)	
31	30	29	28	27 26	25	24	23	22 21	20	19	18	35 18	
0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_ADC_ADC*n*_HOLD Set to 1 to hold the output value on the pad; 0 is for normal operation. (R/W)

RTCIO_ADC_ADC*n*_MUX_SEL 0: route pad to the digital IO_MUX; (R/W)
1: route pad to the RTC block.

RTCIO_ADC_ADC*n*_FUN_SEL Select the RTC function for this pad. 0: select Function 0; 1: select Function 1. (R/W)

RTCIO_ADC_ADC*n*_SLP_SEL Signal selection of pad's sleep mode. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_ADC_ADC*n*_SLP_IE Input enable of the pad in sleep mode. 1 enabled; 0 disabled. (R/W)

RTCIO_ADC_ADC*n*_FUN_IE Input enable of the pad. 1 enabled; 0 disabled. (R/W)

Register 4.49: RTCIO_PAD_DAC1_REG (0x0021)

RTCIO_PAD_PDAC1_DRV	RTCIO_PAD_PDAC1_HOLD	RTCIO_PAD_PDAC1_RDE	RTCIO_PAD_PDAC1_RUE	RTCIO_PAD_PDAC1_DAC	RTCIO_PAD_PDAC1_XPD_DAC	RTCIO_PAD_PDAC1_MUX_SEL	RTCIO_PAD_PDAC1_FUN_SEL	RTCIO_PAD_PDAC1_SLP_SEL	RTCIO_PAD_PDAC1_SLP_IE	RTCIO_PAD_PDAC1_SLP_OE	RTCIO_PAD_PDAC1_FUN_IE	RTCIO_PAD_PDAC1_DAC_XPD_FORCE	(reserved)	
31 30	29	28	27	26 19	18	17	16 15	14	13	12	11	10	19 10	
2	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_PAD_PDAC1_DRV Select the drive strength of the pad. (R/W)

RTCIO_PAD_PDAC1_HOLD Set to 1 to hold the output value on the pad; set to 0 for normal operation. (R/W)

RTCIO_PAD_PDAC1_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_PAD_PDAC1_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_PAD_PDAC1_DAC PAD DAC1 output value. (R/W)

RTCIO_PAD_PDAC1_XPD_DAC Power on DAC1. Usually, PDAC1 needs to be tristated if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

RTCIO_PAD_PDAC1_MUX_SEL 0: route pad to the digital IO_MUX; (R/W)
1: route to the RTC block.

RTCIO_PAD_PDAC1_FUN_SEL the functional selection signal of the pad. (R/W)

RTCIO_PAD_PDAC1_SLP_SEL Sleep mode selection signal of the pad. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_PAD_PDAC1_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC1_SLP_OE Output enable of the pad. 1: enabled ; 0: disabled. (R/W)

RTCIO_PAD_PDAC1_FUN_IE Input enable of the pad. 1: enabled it; 0: disabled. (R/W)

RTCIO_PAD_PDAC1_DAC_XPD_FORCE Power on DAC1. Usually, we need to tristate PDAC1 if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

Register 4.50: RTCIO_PAD_DAC2_REG (0x0022)

RTCIO_PAD_PDAC2_DRV	RTCIO_PAD_PDAC2_HOLD	RTCIO_PAD_PDAC2_RDE	RTCIO_PAD_PDAC2_RUE	RTCIO_PAD_PDAC2_DAC	RTCIO_PAD_PDAC2_XPD_DAC	RTCIO_PAD_PDAC2_MUX_SEL	RTCIO_PAD_PDAC2_FUN_SEL	RTCIO_PAD_PDAC2_SLP_SEL	RTCIO_PAD_PDAC2_SLP_IE	RTCIO_PAD_PDAC2_SLP_OE	RTCIO_PAD_PDAC2_FUN_IE	RTCIO_PAD_PDAC2_DAC_XPD_FORCE	(reserved)	
31 30	29	28	27	26 19	18	17	16 15	14	13	12	11	10	19 10	
2	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_PAD_PDAC2_DRV Select the drive strength of the pad. (R/W)

RTCIO_PAD_PDAC2_HOLD Set to 1 to hold the output value on the pad; 0 is for normal operation. (R/W)

RTCIO_PAD_PDAC2_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_PAD_PDAC2_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_PAD_PDAC2_DAC PAD DAC2 output value. (R/W)

RTCIO_PAD_PDAC2_XPD_DAC Power on DAC2. PDAC2 needs to be tristated if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

RTCIO_PAD_PDAC2_MUX_SEL 0: route pad to the digital IO_MUX; (R/W)
1: route to the RTC block.

RTCIO_PAD_PDAC2_FUN_SEL Select the RTC function for this pad. 0: select Function 0; 1: select Function 1. (R/W)

RTCIO_PAD_PDAC2_SLP_SEL Sleep mode selection signal of the pad. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_PAD_PDAC2_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC2_SLP_OE Output enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC2_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC2_DAC_XPD_FORCE Power on DAC2. Usually, we need to tristate PDAC2 if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

Register 4.51: RTCIO_XTAL_32K_PAD_REG (0x0023)

RTCIO_XTAL_X32N_DRV	RTCIO_XTAL_X32N_HOLD	RTCIO_XTAL_X32N_RDE	RTCIO_XTAL_X32N_RUE	RTCIO_XTAL_X32P_DRV	RTCIO_XTAL_X32P_HOLD	RTCIO_XTAL_X32P_RDE	RTCIO_XTAL_X32P_RUE	RTCIO_XTAL_DAC_XTAL_32K	RTCIO_XTAL_XPD_XTAL_32K	RTCIO_XTAL_X32N_MUX_SEL	RTCIO_XTAL_X32P_MUX_SEL	RTCIO_XTAL_X32N_FUN_SEL	RTCIO_XTAL_X32N_SLP_SEL	RTCIO_XTAL_X32N_SLP_IE	RTCIO_XTAL_X32N_SLP_OE	RTCIO_XTAL_X32N_FUN_IE	RTCIO_XTAL_X32P_FUN_SEL	RTCIO_XTAL_X32P_SLP_SEL	RTCIO_XTAL_X32P_SLP_IE	RTCIO_XTAL_X32P_SLP_OE	RTCIO_XTAL_X32P_FUN_IE	RTCIO_XTAL_DRES_XTAL_32K	RTCIO_XTAL_DBIAS_XTAL_32K	(reserved)	
31 30	29	28	27	26 25	24	23	22	21 20	19	18	17	16 15	14	13	12	11	10 9	8	7	6	5	4 3	2 1	1	
2	0	0	0	2	0	0	0	0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1 0	0 0	0	Reset

RTCIO_XTAL_X32N_DRV Select the drive strength of the pad. (R/W)

RTCIO_XTAL_X32N_HOLD Set to 1 to hold the output value on the pad; 0 is for normal operation. (R/W)

RTCIO_XTAL_X32N_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_XTAL_X32N_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_XTAL_X32P_DRV Select the drive strength of the pad. (R/W)

RTCIO_XTAL_X32P_HOLD Set to 1 to hold the output value on the pad, 0 is for normal operation. (R/W)

RTCIO_XTAL_X32P_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_XTAL_X32P_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_XTAL_DAC_XTAL_32K 32K XTAL bias current DAC value. (R/W)

RTCIO_XTAL_XPD_XTAL_32K Power up 32 KHz crystal oscillator. (R/W)

RTCIO_XTAL_X32N_MUX_SEL 0: route X32N pad to the digital IO_MUX; 1: route to RTC block. (R/W)

RTCIO_XTAL_X32P_MUX_SEL 0: route X32P pad to the digital IO_MUX; 1: route to RTC block. (R/W)

RTCIO_XTAL_X32N_FUN_SEL Select the RTC function. 0: select function 0; 1: select function 1. (R/W)

RTCIO_XTAL_X32N_SLP_SEL Sleep mode selection. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_XTAL_X32N_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32N_SLP_OE Output enable of the pad. 1: enabled; 0; disabled. (R/W)

RTCIO_XTAL_X32N_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32P_FUN_SEL Select the RTC function. 0: select function 0; 1: select function 1. (R/W)

RTCIO_XTAL_X32P_SLP_SEL Sleep mode selection. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_XTAL_X32P_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32P_SLP_OE Output enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32P_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_DRES_XTAL_32K 32K XTAL resistor bias control. (R/W)

RTCIO_XTAL_DBIAS_XTAL_32K 32K XTAL self-bias reference control. (R/W)

Register 4.52: RTCIO_TOUCH_CFG_REG (0x0024)

RTCIO_TOUCH_XPD_BIAS	RTCIO_TOUCH_DREFH	RTCIO_TOUCH_DREFL	RTCIO_TOUCH_DRANGE	RTCIO_TOUCH_DCUR	(reserved)	
31	30 29	28 27	26 25	24 23	45 23	
0	1 1	0 0	1 1	0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_XPD_BIAS Touch sensor bias power on bit. 1: power on; 0: disabled. (R/W)

RTCIO_TOUCH_DREFH Touch sensor saw wave top voltage. (R/W)

RTCIO_TOUCH_DREFL Touch sensor saw wave bottom voltage. (R/W)

RTCIO_TOUCH_DRANGE Touch sensor saw wave voltage range. (R/W)

RTCIO_TOUCH_DCUR Touch sensor bias current. When BIAS_SLEEP is enabled, this setting is available. (R/W)

Register 4.53: RTCIO_TOUCH_PAD*n*_REG (*n*: 0-9) (0x25+1**n*)

(reserved)	RTCIO_TOUCH_PAD*n*_DAC	RTCIO_TOUCH_PAD*n*_START	RTCIO_TOUCH_PAD*n*_TIE_OPT	RTCIO_TOUCH_PAD*n*_XPD	RTCIO_TOUCH_PAD*n*_TO_GPIO	(reserved)	
31 26	25 23	22	21	20	19	37 19	
0 0 0 0 0 0	0x4	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_PAD*n*_DAC Touch sensor slope control. 3-bit for each touch pad, defaults to 100. (R/W)

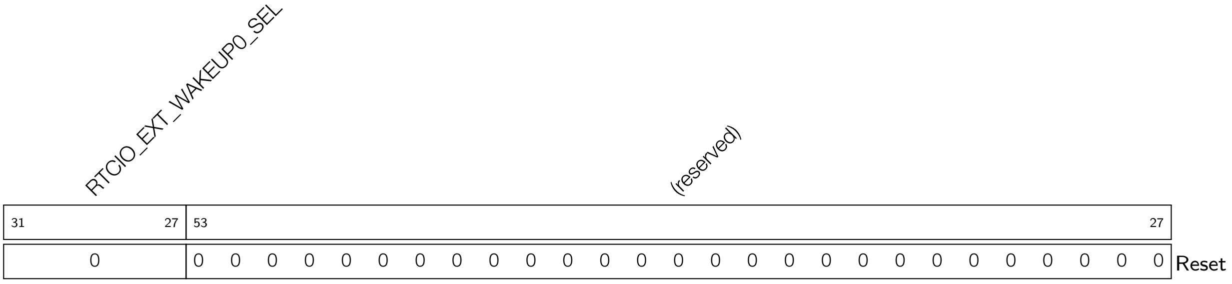
RTCIO_TOUCH_PAD*n*_START Start touch sensor. (R/W)

RTCIO_TOUCH_PAD*n*_TIE_OPT Default touch sensor tie option. 0: tie low; 1: tie high. (R/W)

RTCIO_TOUCH_PAD*n*_XPD Touch sensor power on. (R/W)

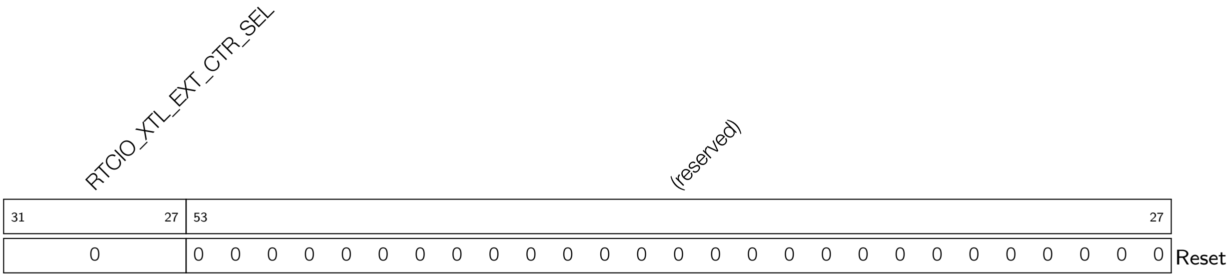
RTCIO_TOUCH_PAD*n*_TO_GPIO Connect the RTC pad input to digital pad input; 0 is available. (R/W)

Register 4.54: RTCIO_EXT_WAKEUP0_REG (0x002f)



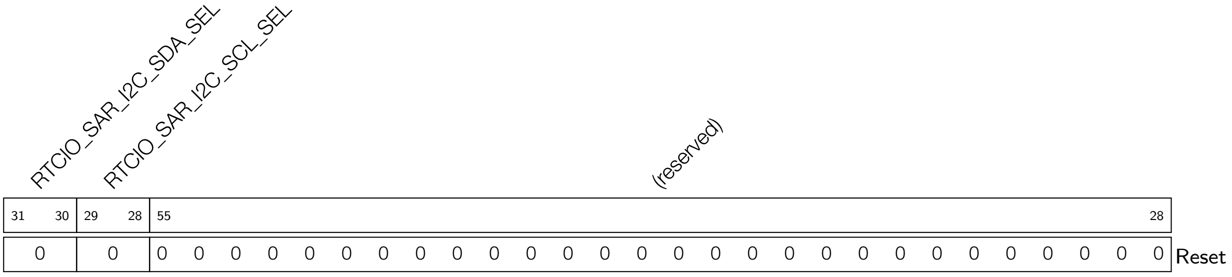
RTCIO_EXT_WAKEUP0_SEL GPIO[0-17] can be used to wake up the chip when the chip is in the sleep mode. This register prompts the pad source to wake up the chip when the latter is in deep/light sleep mode. 0: select GPIO0; 1: select GPIO2, etc. (R/W)

Register 4.55: RTCIO_XTL_EXT_CTR_REG (0x0030)



RTCIO_XTL_EXT_CTR_SEL Select the external crystal power down enable source to get into sleep mode. 0: select GPIO0; 1: select GPIO2, etc. The input value on this pin XOR RTCIO_RTC_EXT_XTAL_CONF_REG[30] is the crystal power down enable signal. (R/W)

Register 4.56: RTCIO_SAR_I2C_IO_REG (0x0031)



RTCIO_SAR_I2C_SDA_SEL Selects a different pad as the RTC I2C SDA signal. 0: use pad TOUCH_PAD[1]; 1: use pad TOUCH_PAD[3]. (R/W)

RTCIO_SAR_I2C_SCL_SEL Selects a different pad as the RTC I2C SCL signal. 0: use pad TOUCH_PAD[1]; 1: use pad TOUCH_PAD[3]. (R/W)

5. I2C Controller

5.1 Overview

An I2C (Inter-Integrated Circuit) bus can be used for communication with several external devices connected to the same bus as ESP32. The ESP32 has dedicated hardware to communicate with peripherals on the I2C bus.

5.2 Features

The I2C controller has the following features:

- Supports both master mode and slave mode
- Supports multi-master and multi-slave communication
- Supports standard mode (100 kbit/s)
- Supports fast mode (400 kbit/s)
- Supports 7-bit addressing and 10-bit addressing
- Supports continuous data transmission with disabled Serial Clock Line (SCL)
- Supports programmable digital noise filter

5.3 Functional Description

5.3.1 Introduction

I2C is a two-wire bus, consisting of an SDA and an SCL line. These lines are configured to open the drain output. The lines are shared by two or more devices, usually one or more masters and one or more slaves.

Communication starts when a master sends out a start condition: it will pull the SDA line low, and will then pull the SCL line high. It will send out nine clock pulses over the SCL line. The first eight pulses are used to shift out a byte, consisting of a 7-bit address and a read/write bit. If a slave with this address is active on the bus, the slave can answer by pulling the SDA low on the ninth clock pulse. The master can now send out more 9-bit clock pulse clusters and, depending on the read/write bit sent, the device or the master will shift out data on the SDA line, with the other side acknowledging the transfer by pulling SDA low on the ninth clock pulse. During data transfer, the SDA line changes only when the SCL line is low. When the master has finished the communication, it will send a stop condition on the bus by raising SDA, while SCL will already be high.

The ESP32 I2C peripheral can handle the I2C protocol, freeing up the processor cores for other tasks.

5.3.2 Architecture

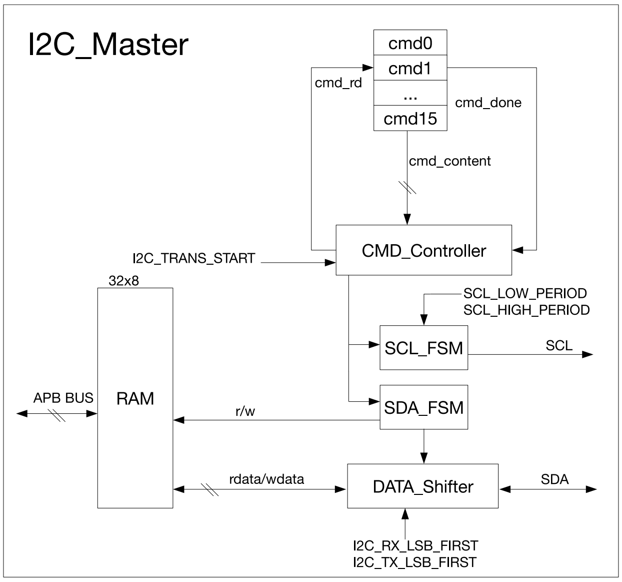


Figure 10: I2C Master Architecture

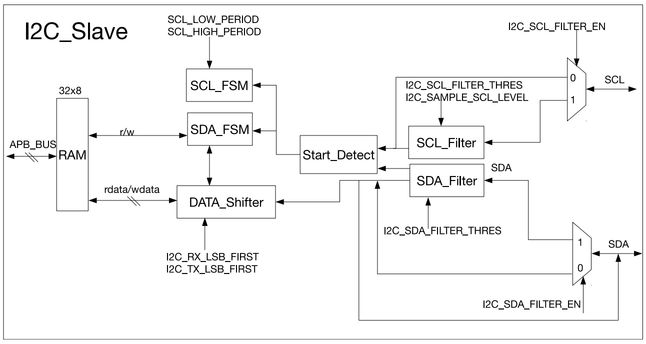


Figure 11: I2C Slave Architecture

An I2C controller can operate either in master mode or slave mode. The I2C_MS_MODE register is used to select the mode. Figure 10 shows the I2C Master architecture, while Figure 11 shows the I2C Slave architecture. The I2C controller contains the following units:

- RAM, the size of which is 32 x 8 bit and it is directly mapped onto the address space of the CPU cores, starting at address REG_I2C_BASE+0x100. Each byte of I2C data is stored in a 32-bit word of memory (so the first byte is at +0x100, the second byte at +0x104, the third byte at +0x108, etc.)
- A CMD_Controller and 16 command registers (cmd0 ~ cmd15), which are used by I2C Master to control data transmission. One command at a time is executed by the I2C controller.
- SCL_FSM: A state machine that controls the SCL clock. The I2C_SCL_HIGH_PERIOD_REG and I2C_SCL_LOW_PERIOD_REG registers are used to configure the frequency and duty cycle of the signal on the SCL line.
- SDA_FSM: A state machine that controls the SDA data line.
- DATA_Shifter which converts the byte data to an outgoing bitstream, or converts an incoming bitstream to byte data. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used for configuring whether the LSB or MSB is stored or transmitted first.

- SCL_Filter and SDA_Filter: Input noise filter for the I2C_Slave. The filter can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN. The filter can remove line glitches with pulse width less than I2C_SCL_FILTER_THRES and I2C_SDA_FILTER_THRES ABP clock cycles.

5.3.3 I2C Bus Timing

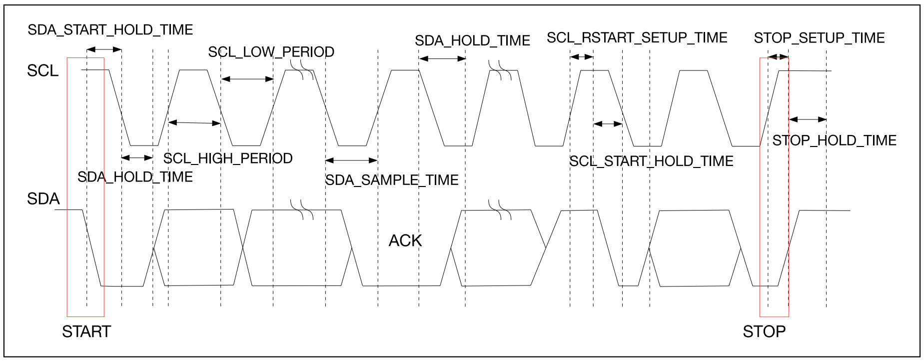


Figure 12: I2C Sequence Chart

Figure 12 is an I2C sequence chart. When the I2C controller works in master mode, SCL is an output signal. In contrast, when the I2C controller works in slave mode, SCL becomes an input signal.

According to the I2C protocol, each transmission of data begins with a START condition and ends with a STOP condition. Data is transmitted by one byte a time, and each byte has an ACK bit. The receiver informs the transmitter to continue transmission by pulling down SDA, which indicates an ACK. The receiver can also indicate it wants to stop the transmission by not pulling down the SDA line, thereby not giving an ACK.

Figure 12 also shows the registers that can configure the START bit, STOP bit, SDA hold time, and SDA sample time.

5.3.4 I2C cmd Structure

	31		13:11	10	9	8	7:0
cmd0	CMD_DONE	...	op_code	ack_value	ack_exp	ack_check_en	byte_num

...

	31		13:11	10	9	8	7:0
cmd15	CMD_DONE	...	op_code	ack_value	ack_exp	ack_check_en	byte_num

Figure 13: Structure of The I2C Command Register

The Command register is active only in I2C master mode, with its internal structure shown in Figure 13.

CMD_DONE: The CMD_DONE bit of every command can be read by software to tell if the command has been handled by hardware.

op_code: op_code is used to indicate the command. The I2C controller supports four commands:

- RSTART: op_code = 0 is the RSTART command to control the transmission of a START or RESTART I2C condition.
- WRITE: op_code = 1 is the WRITE command for the I2C Master to transmit data.

- READ: op_code = 2 is the READ command for the I2C Master to receive data.
- STOP: op_code = 3 is the STOP command to control the transmission of a STOP I2C condition.
- END: op_code = 4 is the END command for continuous data transmission. When the END command is given, SCL is temporarily disabled to allow software to reload the command and data registers for subsequent events before resuming. Transmission will then continue seamlessly.

A complete data transmission process begins with an RSTART command, and ends with a STOP command.

ack_value: When receiving data, this bit is used to indicate whether the receiver will send an ACK after this byte has been received.

ack_exp: This bit is to set an expected ACK value for the transmitter.

ack_check_en: When transmitting a byte, this bit enables checking the ACK value received against the ack_exp value. Checking is enabled by 1, while 0 disables it.

byte_num: This register specifies the length of data to be read or written. When the op_code is RSTART, STOP or END, this value has no meaning.

5.3.5 I2C Master Writes to Slave

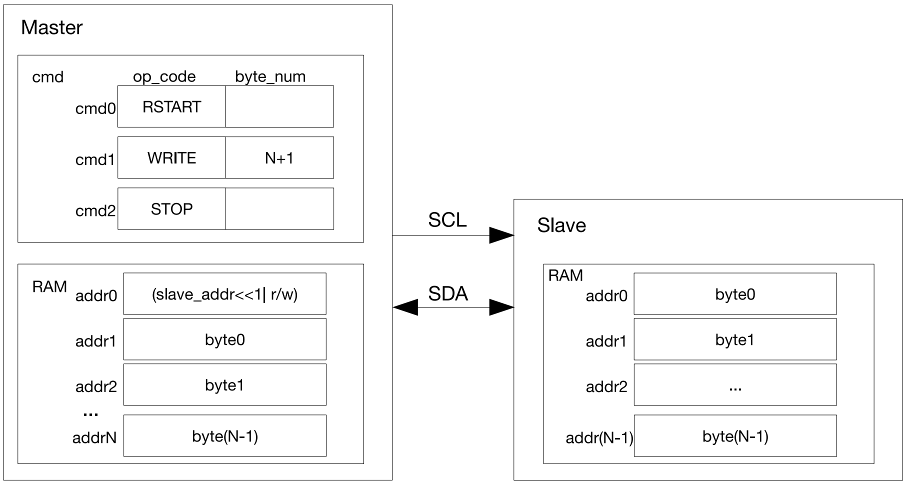


Figure 14: I2C Master Writes to Slave with 7-bit Address

Figure 14 shows the I2C Master writing N bytes of data to an external I2C Slave; both are supposed to be ESP32 I2C controllers. According to the I2C protocol, the first byte is the Slave address. As shown in the diagram, the first byte of the RAM unit has been populated with the Slave's 7-bit address plus the 1-bit read/write flag. In this case, the flag is zero, indicating a write operation. The rest of the RAM unit stores N bytes of data that are ready for transmission. The cmd unit has been populated with the sequence of commands for the operation.

The FIFO offset in RAM can be configured via the TXFIFO_START_ADDR field in the RXFIFO_ST_REG register.

When all registers are ready, the I2C_TRANS_START bit in I2C_CTR_REG is set to start the transmission. Then, the I2C Master initiates a START condition to activate the slave devices. I2C Master will then progress to the WRITE command which will cause N+1 bytes to be fetched from RAM and sent to the Slave. The first of these bytes is the address byte. Each slave device will compare this to its own. If the addresses do not match, the slave will ignore the rest of the transmission. If they do match, the slave will ACK the initial byte and the I2C master will continue sending the rest of the data; when ack_check_en is set to 'one', Master will check ACK value.

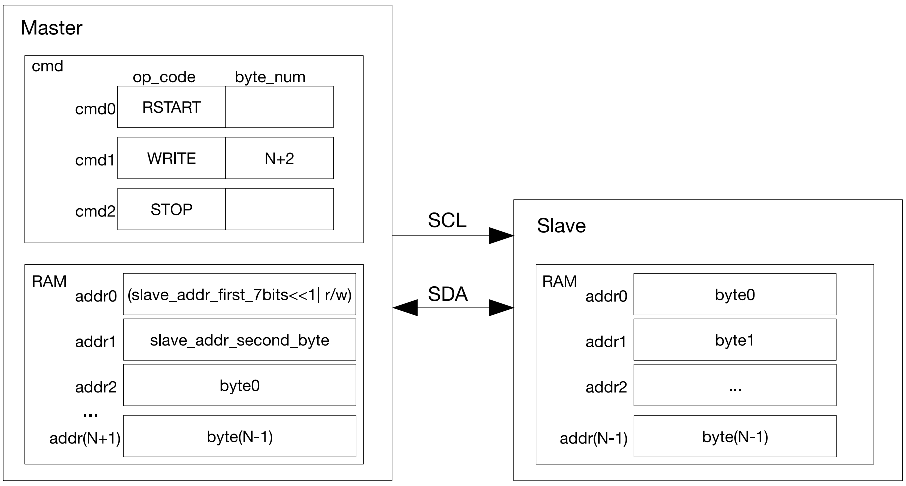


Figure 15: I2C Master Writes to Slave with 10-bit Address

The I2C controller uses 7-bit addressing by default. However, 10-bit addressing can also be used. In the master, this is done by sending a second I2C address byte after the first address byte. In the slave, the I2C_SLAVE_ADDR_10BIT_EN register bit can be set to activate a 10-bit addressing. I2C_SLAVE_ADDR is used to configure I2C Slave's address, as per usual. Figure 15 shows the equivalent of I2C Master operation writing N-bytes of data to an I2C Slave with a 10-bit address. Since 10-bit Slave addresses require an extra address byte, both the byte_num field of the WRITE command and the number of total bytes in RAM increase by one.

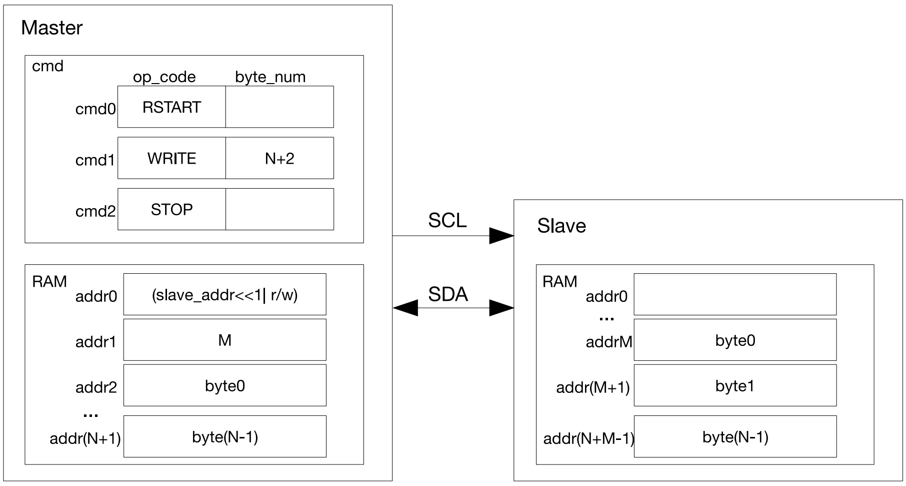


Figure 16: I2C Master Writes to addrM in RAM of Slave with 7-bit Address

One way many I2C Slave devices are designed is by exposing a register block containing various settings. The I2C Master can write one or more of these registers by sending the Slave a register address. The ESP32 I2C Slave controller has hardware support for such a scheme.

Specifically, on the Slave, I2C_FIFO_ADDR_CFG_EN can be set so that the I2C Master can write to a specified register address inside the I2C Slave memory block. Figure 16 shows the I2C Master writing N-bytes of data byte0 ~ byte(N-1) from the RAM unit to register address M (determined by addrM in RAM unit) with the Slave.

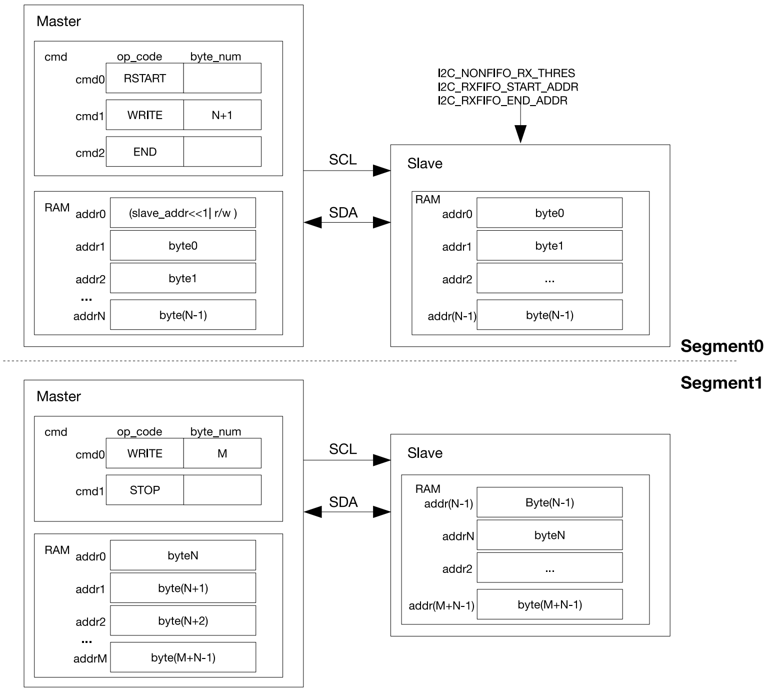


Figure 17: I2C Master Writes to Slave with 7-bit Address in Two Segments

If the data size exceeds the RAM unit capacity of 32 bytes, the END command can be called to enable segmented transmission. Figure 17 shows I2C Master writing data in two segments to Slave. The upper part of the figure shows the configuration of the first sequence of bytes in the transfer. I2C Master will turn off SCL clock, after executing the END command and after the controller generates the I2C_END_DETECT_INT interrupt.

On receiving I2C_END_DETECT_INT (or polling the CMD_DONE bit of the command register the END was placed into), software should refresh the contents of the cmd and RAM units, as shown in the lower part of the figure. Subsequently, it should clear the I2C_END_DETECT_INT interrupt and resume the transaction by setting the I2C_TRANS_START bit in CTR_CTR_REG.

5.3.6 I2C Master Reads from Slave

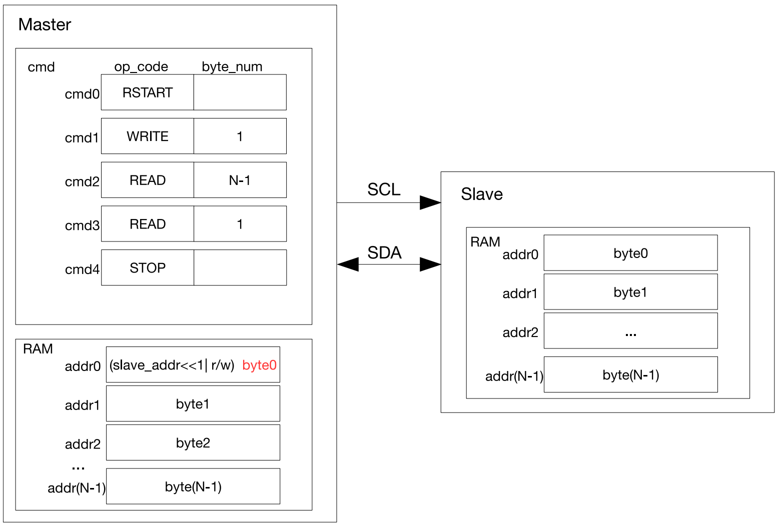


Figure 18: I2C Master Reads from Slave with 7-bit Address

Figure 18 shows the I2C Master reading N-bytes of data from an I2C Slave with a 7-bit address. At first, the I2C Master needs to send the address of the I2C Slave, so cmd1 is a WRITE command. The byte that this command sends is the I2C slave address plus the R/W flag, which in this case is 1 and, therefore, indicates that this is going to be a read operation. According to the I2C protocol, I2C Master will not return ACK on receiving the last byte of data read from the slave; consequently, READ is divided into two segments. The I2C Master replies ACK to N-1 bytes in cmd2 and does not reply ACK to the single byte READ command in cmd3, i.e., the last transmitted data.

When storing the received data, I2C Master will start from the first address in RAM. Byte0 (Slave address + 1-bit R/W marker bit) will be overwritten. The FIFO RAM offsets reading and writing data which can then be configured via the RXFIFO_START_ADDR and TXFIFO_START_ADDR fields in the RXFIFO_ST_REG register.

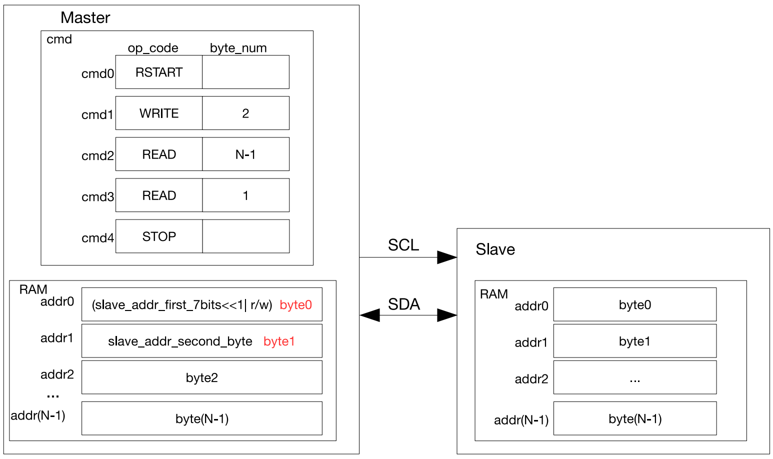


Figure 19: I2C Master Reads from Slave with 10-bit Address

Figure 19 shows the I2C Master reading data from a slave with a 10-bit address. In the Slave, this mode is enabled by setting I2C_SLAVE_ADDR_10BIT_EN register. In the Master, two bytes of RAM are used for a 10-bit address.

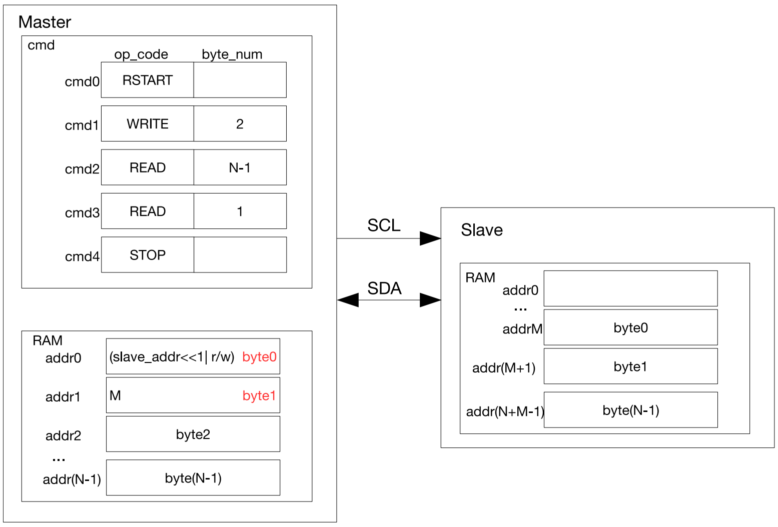


Figure 20: I2C Master Reads N Bytes of Data from addrM in Slave with 7-bit Address

Figure 20 shows the I2C Master selecting a register address inside the I2C Slave device and then reading data from it and subsequent addresses. This mode is enabled by setting the I2C_FIFO_ADDR_CFG_EN register in the Slave. The internal register address of the Slave, M, is stored in the RAM byte following the address. The WRITE command has a length of two data bytes to compensate for this.

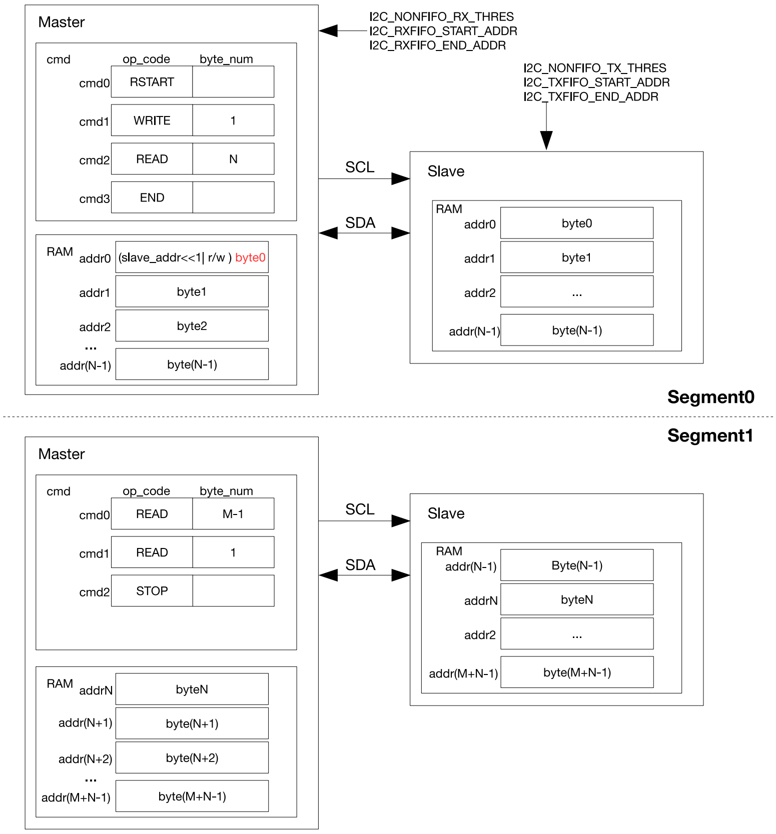


Figure 21: I2C Master Reads from Slave with 7-bit Address in Two Segments

Figure 21 shows the I2C Master reading N+M bytes of data in two segments from I2C Slave by using the END command. This allows for more data to be read than what can be fitted into the RAM. The upper part of the figure shows the configuration of Segment0. The Master will update the configuration of cmd after executing the END command, as shown in the lower part of the figure. I2C Slave will refresh the data before its RAM is empty.

5.3.7 Interrupts

- I2C_TX_SEND_EMPTY_INT: Triggered when I2C sends more data than nonfifo_tx_thres.
- I2C_RX_REC_FULL_INT: Triggered when I2C receives more data than nonfifo_rx_thres.
- I2C_ACK_ERR_INT: Triggered when I2C receives a wrong ACK bit..
- I2C_TRANS_START_INT: Triggered when I2C sends the START bit.
- I2C_TIME_OUT_INT: Triggered when I2C takes too long to receive data.
- I2C_TRANS_COMPLETE_INT: Triggered when I2C Master has finished STOP command or when I2C Slave detects STOP bit.
- I2C_MASTER_TRAN_COMP_INT: Triggered when I2C Master sends or receives a byte.
- I2C_ARBITRATION_LOST_INT: Triggered when I2C Master has lost the usage right of I2C Bus.
- I2C_END_DETECT_INT: Triggered when I2C deals with the END command.

5.4 Register Summary

Name	Description	I2C0	I2C1	Acc
Configuration registers				
I2C_SLAVE_ADDR_REG	Configures the I2C slave address	0x3FF53010	0x3FF67010	R/W
I2C_RXFIFO_ST_REG	FIFO status register	0x3FF53014	0x3FF67014	RO
I2C_FIFO_CONF_REG	FIFO configuration register	0x3FF53018	0x3FF67018	R/W
Timing registers				
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x3FF53030	0x3FF67030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x3FF53034	0x3FF67034	R/W
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL clock	0x3FF53000	0x3FF67000	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of the SCL clock	0x3FF53038	0x3FF67038	R/W
I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x3FF53040	0x3FF67040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x3FF53044	0x3FF67044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x3FF53048	0x3FF67048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x3FF5304C	0x3FF6704C	R/W
Filter registers				
I2C_SCL_FILTER_CFG_REG	SCL filter configuration register	0x3FF53050	0x3FF67050	R/W
I2C_SDA_FILTER_CFG_REG	SDA filter configuration register	0x3FF53054	0x3FF67054	R/W
Interrupt registers				
I2C_INT_RAW_REG	Raw interrupt status	0x3FF53020	0x3FF67020	RO
I2C_INT_ENA_REG	Interrupt enable bits	0x3FF53028	0x3FF67028	R/W
I2C_INT_CLR_REG	Interrupt clear bits	0x3FF53024	0x3FF67024	WO
Command registers				
I2C_COMD0_REG	I2C command register 0	0x3FF53058	0x3FF67058	R/W
I2C_COMD1_REG	I2C command register 1	0x3FF5305C	0x3FF6705C	R/W
I2C_COMD2_REG	I2C command register 2	0x3FF53060	0x3FF67060	R/W
I2C_COMD3_REG	I2C command register 3	0x3FF53064	0x3FF67064	R/W
I2C_COMD4_REG	I2C command register 4	0x3FF53068	0x3FF67068	R/W
I2C_COMD5_REG	I2C command register 5	0x3FF5306C	0x3FF6706C	R/W
I2C_COMD6_REG	I2C command register 6	0x3FF53070	0x3FF67070	R/W
I2C_COMD7_REG	I2C command register 7	0x3FF53074	0x3FF67074	R/W
I2C_COMD8_REG	I2C command register 8	0x3FF53078	0x3FF67078	R/W
I2C_COMD9_REG	I2C command register 9	0x3FF5307C	0x3FF6707C	R/W
I2C_COMD10_REG	I2C command register 10	0x3FF53080	0x3FF67080	R/W
I2C_COMD11_REG	I2C command register 11	0x3FF53084	0x3FF67084	R/W
I2C_COMD12_REG	I2C command register 12	0x3FF53088	0x3FF67088	R/W

Name	Description	I2C0	I2C1	Acc
I2C_COMD13_REG	I2C command register 13	0x3FF5308C	0x3FF6708C	R/W
I2C_COMD14_REG	I2C command register 14	0x3FF53090	0x3FF67090	R/W
I2C_COMD15_REG	I2C command register 15	0x3FF53094	0x3FF67094	R/W

5.5 Registers

Register 5.1: I2C_SCL_LOW_PERIOD_REG (0x0000)

(reserved)	I2C_SCL_LOW_PERIOD	
31 … 14	13 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_LOW_PERIOD This register is used to configure the low-level width of the SCL clock signal, in APB clock cycles. (R/W)

Register 5.2: I2C_CTR_REG (0x0004)

(reserved)	I2C_RX_LSB_FIRST	I2C_TX_LSB_FIRST	I2C_TRANS_START	I2C_MS_MODE	(reserved)	I2C_SAMPLE_SCL_LEVEL	I2C_SCL_FORCE_OUT	I2C_SDA_FORCE_OUT	
31 … 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	1	1	Reset

I2C_RX_LSB_FIRST This bit is used to control the storage mode for received data. (R/W)
1: receive data from the least significant bit;
0: receive data from the most significant bit.

I2C_TX_LSB_FIRST This bit is used to control the sending mode for data needing to be sent. (R/W)
1: send data from the least significant bit;
0: send data from the most significant bit.

I2C_TRANS_START Set this bit to start sending the data in txfifo. (R/W)

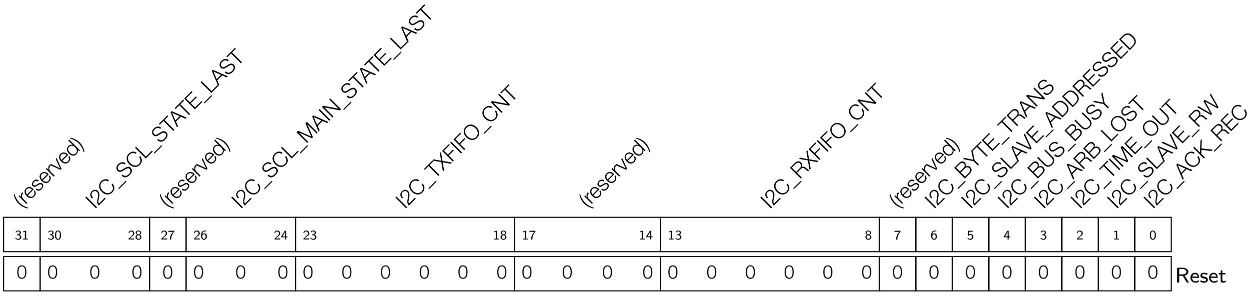
I2C_MS_MODE Set this bit to configure the module as an I2C Master. Clear this bit to configure the module as an I2C Slave. (R/W)

I2C_SAMPLE_SCL_LEVEL 1: sample SDA data on the SCL low level; 0: sample SDA data on the SCL high level. (R/W)

I2C_SCL_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

I2C_SDA_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

Register 5.3: I2C_SR_REG (0x0008)



I2C_TXFIFO_CNT This field stores the amount of received data in RAM. (RO)

I2C_RXFIFO_CNT This field represents the amount of data needed to be sent. (RO)

I2C_BYTE_TRANS This field changes to 1 when one byte is transferred. (RO)

I2C_SLAVE_ADDRESSED When configured as an I2C Slave, and the address sent by the master is equal to the address of the slave, then this bit will be of high level. (RO)

I2C_BUS_BUSY 1: the I2C bus is busy transferring data; 0: the I2C bus is in idle state. (RO)

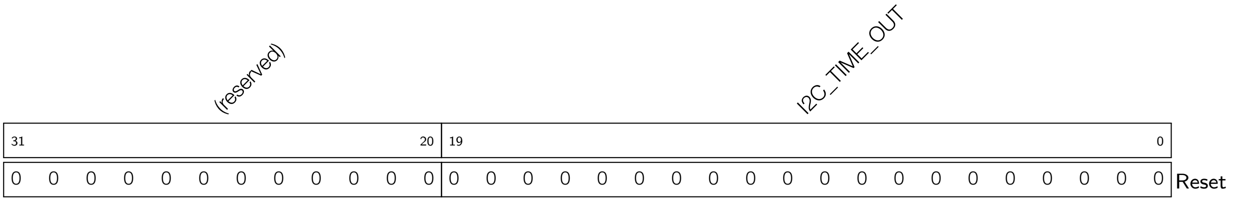
I2C_ARB_LOST When the I2C controller loses control of SCL line, this register changes to 1. (RO)

I2C_TIME_OUT When the I2C controller takes more than I2C_TIME_OUT clocks to receive a data bit, this field changes to 1. (RO)

I2C_SLAVE_RW When in slave mode, 1: master reads from slave; 0: master writes to slave. (RO)

I2C_ACK_REC This register stores the value of the received ACK bit. (RO)

Register 5.4: I2C_TO_REG (0x000c)



I2C_TIME_OUT This register is used to configure the timeout for receiving a data bit in APB clock cycles. (R/W)

Register 5.5: I2C_SLAVE_ADDR_REG (0x0010)

I2C_SLAVE_ADDR_10BIT_EN	(reserved)	I2C_SLAVE_ADDR	
31	30 15	14 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SLAVE_ADDR_10BIT_EN This field is used to enable the slave 10-bit addressing mode. (R/W)

I2C_SLAVE_ADDR When configured as an I2C Slave, this field is used to configure the slave address. (R/W)

Register 5.6: I2C_RXFIFO_ST_REG (0x0014)

(reserved)	I2C_RXFIFO_TXFIFO_END_ADDR	I2C_RXFIFO_TXFIFO_START_ADDR	I2C_RXFIFO_END_ADDR	I2C_RXFIFO_START_ADDR	
31 20	19 15	14 10	9 5	4 0	
0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	Reset

I2C_TXFIFO_END_ADDR This is the offset address of the last sent data, as described in nonfifo_tx_thres register. (RO)

I2C_TXFIFO_START_ADDR This is the offset address of the first sent data, as described in nonfifo_tx_thres register. (RO)

I2C_RXFIFO_END_ADDR This is the offset address of the first received data, as described in nonfifo_rx_thres_register. (RO)

I2C_RXFIFO_START_ADDR This is the offset address of the last received data, as described in nonfifo_rx_thres_register. (RO)

Register 5.7: I2C_FIFO_CONF_REG (0x0018)

(reserved)	I2C_NONFIFO_TX_THRES	I2C_NONFIFO_RX_THRES	(reserved)	I2C_FIFO_ADDR_CFG_EN	I2C_NONFIFO_EN	
31 26	25 20	19 14	13 12	11	10	
0 0 0 0 0 0	0x15	0x15	0 0	0	0	Reset

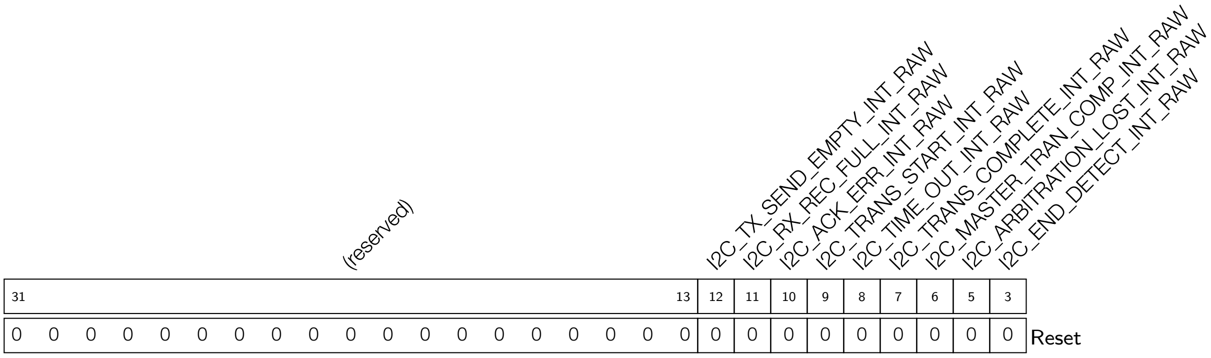
I2C_NONFIFO_TX_THRES When I2C sends more than nonfifo_tx_thres bytes of data, it will generate a tx_send_empty_int_raw interrupt and update the current offset address of the sent data. (R/W)

I2C_NONFIFO_RX_THRES When I2C receives more than nonfifo_rx_thres bytes of data, it will generate a rx_send_full_int_raw interrupt and update the current offset address of the received data. (R/W)

I2C_FIFO_ADDR_CFG_EN When this bit is set to 1, the byte received after the I2C address byte represents the offset address in the I2C Slave RAM. (R/W)

I2C_NONFIFO_EN Set this bit to enble APB nonfifo access. (R/W)

Register 5.8: I2C_INT_RAW_REG (0x0020)



I2C_TX_SEND_EMPTY_INT_RAW The raw interrupt status bit for the I2C_TX_SEND_EMPTY_INT interrupt. (RO)

I2C_RX_REC_FULL_INT_RAW The raw interrupt status bit for the I2C_RX_REC_FULL_INT interrupt. (RO)

I2C_ACK_ERR_INT_RAW The raw interrupt status bit for the I2C_ACK_ERR_INT interrupt. (RO)

I2C_TRANS_START_INT_RAW The raw interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_TIME_OUT_INT_RAW The raw interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_MASTER_TRAN_COMP_INT_RAW The raw interrupt status bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_END_DETECT_INT_RAW The raw interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

Register 5.9: I2C_INT_CLR_REG (0x0024)

(reserved)	I2C_TX_SEND_EMPTY_INT_CLR	I2C_RX_REC_FULL_INT_CLR	I2C_ACK_ERR_INT_CLR	I2C_TRANS_START_INT_CLR	I2C_TIME_OUT_INT_CLR	I2C_TRANS_COMPLETE_INT_CLR	I2C_MASTER_TRAN_COMP_INT_CLR	I2C_ARBITRATION_LOST_INT_CLR	I2C_END_DETECT_INT_CLR	
31 13	12	11	10	9	8	7	6	5	3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

I2C_TX_SEND_EMPTY_INT_CLR Set this bit to clear the I2C_TX_SEND_EMPTY_INT interrupt. (WO)

I2C_RX_REC_FULL_INT_CLR Set this bit to clear the I2C_RX_REC_FULL_INT interrupt. (WO)

I2C_ACK_ERR_INT_CLR Set this bit to clear the I2C_ACK_ERR_INT interrupt. (WO)

I2C_TRANS_START_INT_CLR Set this bit to clear the I2C_TRANS_START_INT interrupt. (WO)

I2C_TIME_OUT_INT_CLR Set this bit to clear the I2C_TIME_OUT_INT interrupt. (WO)

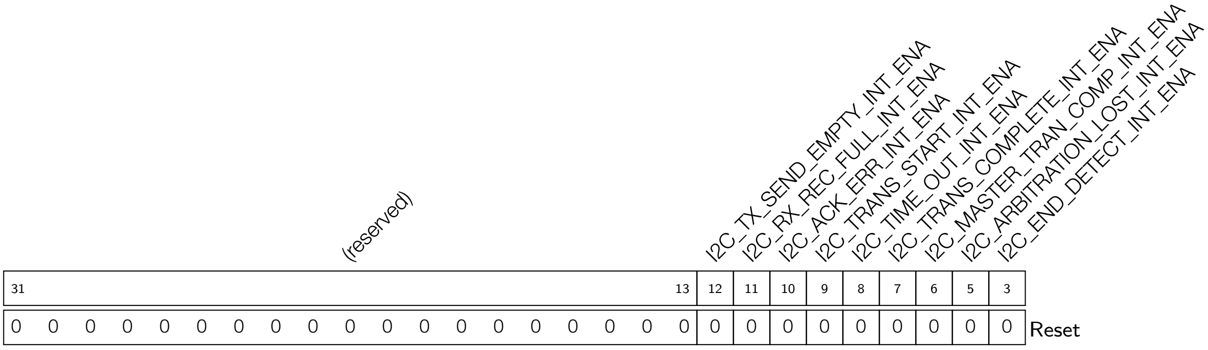
I2C_TRANS_COMPLETE_INT_CLR Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WO)

I2C_MASTER_TRAN_COMP_INT_CLR Set this bit to clear the I2C_MASTER_TRAN_COMP_INT interrupt. (WO)

I2C_ARBITRATION_LOST_INT_CLR Set this bit to clear the I2C_ARBITRATION_LOST_INT interrupt. (WO)

I2C_END_DETECT_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

Register 5.10: I2C_INT_ENA_REG (0x0028)



I2C_TX_SEND_EMPTY_INT_ENA The interrupt enable bit for the I2C_TX_SEND_EMPTY_INT interrupt. (R/W)

I2C_RX_REC_FULL_INT_ENA The interrupt enable bit for the I2C_RX_REC_FULL_INT interrupt. (R/W)

I2C_ACK_ERR_INT_ENA The interrupt enable bit for the I2C_ACK_ERR_INT interrupt. (R/W)

I2C_TRANS_START_INT_ENA The interrupt enable bit for the I2C_TRANS_START_INT interrupt. (R/W)

I2C_TIME_OUT_INT_ENA The interrupt enable bit for the I2C_TIME_OUT_INT interrupt. (R/W)

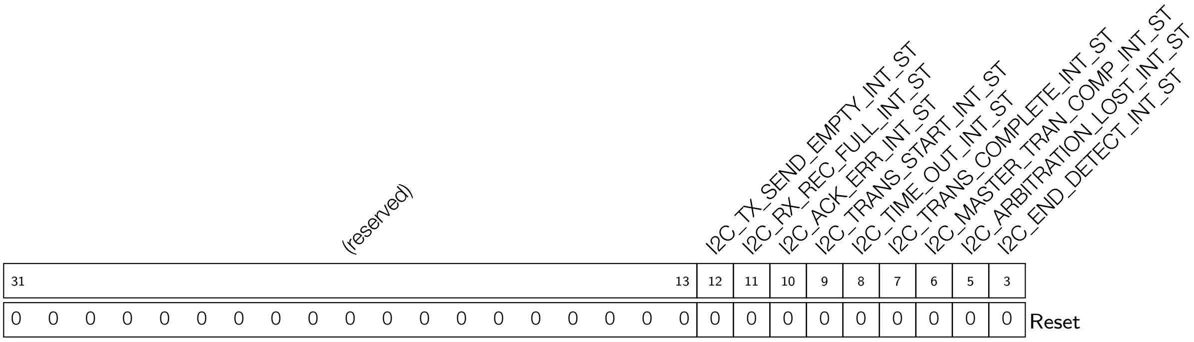
I2C_TRANS_COMPLETE_INT_ENA The interrupt enable bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_MASTER_TRAN_COMP_INT_ENA The interrupt enable bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (R/W)

I2C_ARBITRATION_LOST_INT_ENA The interrupt enable bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

I2C_END_DETECT_INT_ENA The interrupt enable bit for the I2C_END_DETECT_INT interrupt. (R/W)

Register 5.11: I2C_INT_STATUS_REG (0x002c)



I2C_TX_SEND_EMPTY_INT_ST The masked interrupt status bit for the I2C_TX_SEND_EMPTY_INT interrupt. (RO)

I2C_RX_REC_FULL_INT_ST The masked interrupt status bit for the I2C_RX_REC_FULL_INT interrupt. (RO)

I2C_ACK_ERR_INT_ST The masked interrupt status bit for the I2C_ACK_ERR_INT interrupt. (RO)

I2C_TRANS_START_INT_ST The masked interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_TIME_OUT_INT_ST The masked interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

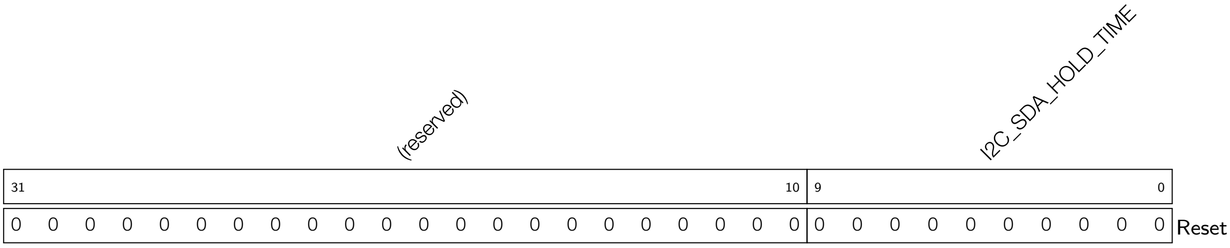
I2C_TRANS_COMPLETE_INT_ST The masked interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_MASTER_TRAN_COMP_INT_ST The masked interrupt status bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_END_DETECT_INT_ST The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

Register 5.12: I2C_SDA_HOLD_REG (0x0030)



I2C_SDA_HOLD_TIME This register is used to configure the time to hold the data after the negative edge of SCL, in APB clock cycles. (R/W)

Register 5.13: I2C_SDA_SAMPLE_REG (0x0034)

(reserved)	I2C_SDA_SAMPLE_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	Reset

I2C_SDA_SAMPLE_TIME This register is used to configure the delay between the positive edge of SCL and the I2C controller sampling SDA, in APB clock cycles. (R/W)

Register 5.14: I2C_SCL_HIGH_PERIOD_REG (0x0038)

(reserved)	I2C_SCL_HIGH_PERIOD	
31 14	13 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_HIGH_PERIOD This register is used to configure how long SCL is kept high, in APB clock cycles. (R/W)

Register 5.15: I2C_SCL_START_HOLD_REG (0x0040)

(reserved)	I2C_SCL_START_HOLD_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0	Reset

I2C_SCL_START_HOLD_TIME This register is used to configure the time between the negative edge of SDA and the negative edge of SCL for a START condition, in APB clock cycles. (R/W)

Register 5.16: I2C_SCL_RSTART_SETUP_REG (0x0044)

(reserved)	I2C_SCL_RSTART_SETUP_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0	Reset

I2C_SCL_RSTART_SETUP_TIME This register is used to configure the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition, in APB clock cycles. (R/W)

Register 5.17: I2C_SCL_STOP_HOLD_REG (0x0048)

(reserved)	I2C_SCL_STOP_HOLD_TIME	
31 14	13 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_STOP_HOLD_TIME This register is used to configure the delay after the STOP condition's positive edge, in APB clock cycles. (R/W)

Register 5.18: I2C_SCL_STOP_SETUP_REG (0x004C)

(reserved)	I2C_SCL_STOP_SETUP_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_STOP_SETUP_TIME This register is used to configure the time between the positive edge of SCL and the positive edge of SDA, in APB clock cycles. (R/W)

Register 5.19: I2C_SCL_FILTER_CFG_REG (0x0050)

(reserved)	I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	
31 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	Reset

I2C_SCL_FILTER_EN This is the filter enable bit for SCL. (R/W)

I2C_SCL_FILTER_THRES When a pulse on the SCL input has smaller width than this register value in APB clock cycles, the I2C controller will ignore that pulse. (R/W)

Register 5.20: I2C_SDA_FILTER_CFG_REG (0x0054)

(reserved)	I2C_SDA_FILTER_EN	I2C_SDA_FILTER_THRES	
31 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	Reset

I2C_SDA_FILTER_EN This is the filter enable bit for SDA. (R/W)

I2C_SDA_FILTER_THRES When a pulse on the SDA input has smaller width than this register value in APB clock cycles, the I2C controller will ignore that pulse. (R/W)

Register 5.21: I2C_COMD*n*_REG (*n*: 0-15) (0x58+4**n*)

I2C_COMMAND*n*_DONE	(reserved)	I2C_COMMAND*n*	
31	30 14	13 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_COMMAND*n*_DONE When command *n* is done in I2C Master mode, this bit changes to high level. (R/W)

I2C_COMMAND*n* This is the content of command *n*. It consists of three parts: (R/W)
op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END.
Byte_num represents the number of bytes that need to be sent or received.
ack_check_en, ack_exp and ack are used to control the ACK bit. See I2C cmd structure for more information.

6. LED_PWM

6.1 Introduction

The LED_PWM controller is primarily designed to control the intensity of LEDs, although it can be used to generate PWM signals for other purposes as well. It has 16 channels which can generate independent waveforms that can be used to drive RGB LED devices. For maximum flexibility, the high-speed as well as the low-speed channels can be driven from one of four high-speed/low-speed timers. The PWM controller also has the ability to automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference. To increase resolution, the LED_PWM controller is also able to dither between two values, when a fractional PWM value is configured.

The LED_PWM controller has eight high-speed and eight low-speed PWM generators. In this document, they will be referred to as hsch*n* and lsch*n*, respectively. These channels can be driven from four timers which will be indicated by h_timer*x* and l_timer*x*.

6.2 Functional Description

6.2.1 Architecture

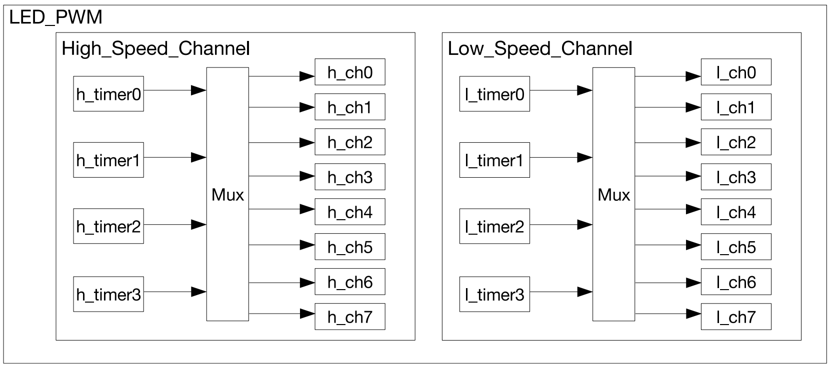


Figure 22: LED_PWM Architecture

Figure 22 shows the architecture of the LED_PWM controller. As can be seen in the figure, the LED_PWM controller contains eight high-speed and eight low-speed channels. There are four high-speed clock modules for the high-speed channels, from which one h_timer*x* can be selected. There are also four low-speed clock modules for the low-speed channels, from which one l_timer*x* can be selected.

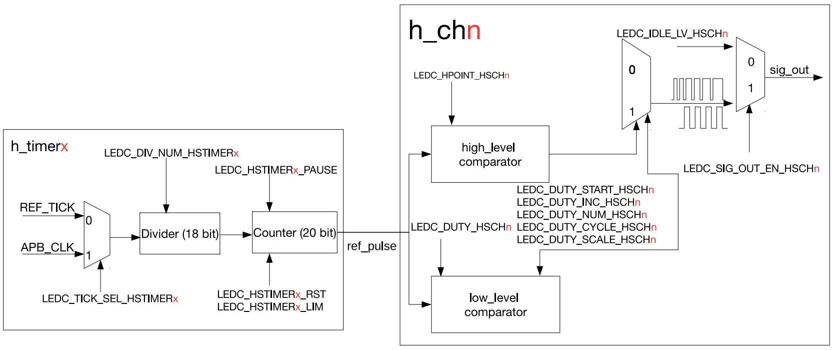


Figure 23: LED_PWM High-speed Channel Diagram

Figure 23 illustrates a PWM channel with its selected timer; in this instance a high-speed channel and associated high-speed timer.

6.2.2 Timers

A high-speed timer consists of a multiplexer to select one of two clock sources: either REF_TICK or APB_CLK. For more information on the clock sources, please see Chapter Reset And Clock. The input clock is divided down by a divider first. The division factor is specified by LEDC_DIV_NUM_HSTIMERx which contains a fixed point number: the highest 10 bits represent the integer portion, while the lowest eight bits contain the fractional portion.

The divided clock signal is then fed into a 20-bit counter. This counter will count up to the value specified in LEDC_HSTIMERx_LIM. An overflow interrupt will be generated once the counting value reaches this limit, at which point the counter restarts counting from zero. It is also possible to reset, suspend, and read the values of the counter by software.

The output signal of the timer is the 20-bit value generated by the counter. The cycle period of this signal defines the frequency of the signals of any PWM channels connected to this timer. This frequency depends on both the division factor of the divider, as well as the range of the counter:

$$f_{\text{sig\_out}} = \frac{f_{\text{REF\_TICK}} \cdot (!\text{LEDC\_TICK\_SEL\_HSTIMER}x) + f_{\text{APB\_CLK}} \cdot \text{LEDC\_TICK\_SEL\_HSTIMER}x}{\text{LEDC\_DIV\_NUM\_HSTIMER}x \cdot 2^{\text{LEDC\_HSTIMER}x\text{\_LIM}}}$$

The low-speed timers l_timerx on the low-speed channel differ from the high-speed timers h_timerx in two aspects:

1. Where the high-speed timer clock source can be clocked from REF_TICK or APB_CLK, the low speed timers are sourced from either REF_TICK or SLOW_CLOCK. The SLOW_CLOCK source can be either APB_CLK (80 MHz) or 8 MHz, and can be selected using LEDC_APB_CLK_SEL.
2. The high-speed counter and divider are glitch-free, which means that if the software modifies the maximum counter or divisor value, the update will come into effect after the next overflow interrupt. In contrast, the low-speed counter and divider will update these values only when LEDC_LSTIMERx_PARA_UP is set.

6.2.3 Channels

A channel takes the 20-bit value from the counter of the selected high-speed timer and compares it to a set of two values in order to set the channel output. The first value it is compared to is the content of LEDC_HPOINT_HSCHn; if these two match, the output will be latched high. The second value is the sum of LEDC_HPOINT_HSCHn and LEDC_DUTY_HSCHn[24..4]. When this value is reached, the output is latched low. By using these two values, the relative phase and the duty cycle of the PWM output can be set. Figure 24 illustrates this.

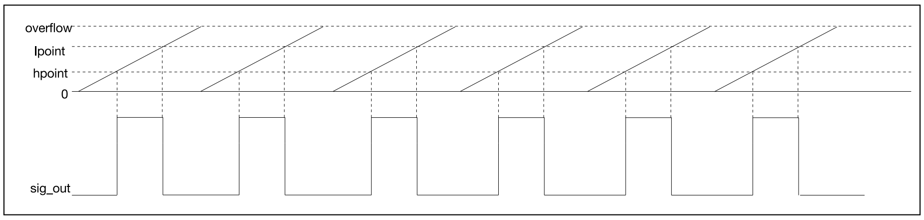


Figure 24: LED PWM Output Signal Diagram

LEDC_DUTY_HSCH*n* is a fixed-point register with four fractional bits. As mentioned before, when LEDC_DUTY_HSCH*n*[24..4] is used in the PWM calculation directly, LEDC_DUTY_HSCH*n*[3..0] can be used to dither the output. If this value is non-zero, with a statistical chance of LEDC_DUTY_HSCH*n*[3..0]/16, the actual PWM pulse will be one cycle longer. This effectively increases the resolution of the PWM generator to 24 bits, but at the cost of a slight jitter in the duty cycle.

The channels also have the ability to automatically fade from one duty cycle value to another. This feature is enabled by setting LEDC_DUTY_START_HSCH*n*. When this bit is set, the PWM controller will automatically increment or decrement the value in LEDC_DUTY_HSCH*n*, depending on whether the bit LEDC_DUTY_INC_HSCH*n* is set or cleared, respectively. The speed the duty cycle changes is defined as such: every time the LEDC_DUTY_CYCLE_HSCH*n* cycles, the content of LEDC_DUTY_SCALE_HSCH*n* is added to or subtracted from LEDC_DUTY_HSCH*n*[24..4]. The length of the fade can be limited by setting LEDC_DUTY_NUM_HSCH*n*: the fade will only last that number of cycles before finishing. A finished fade also generates an interrupt.

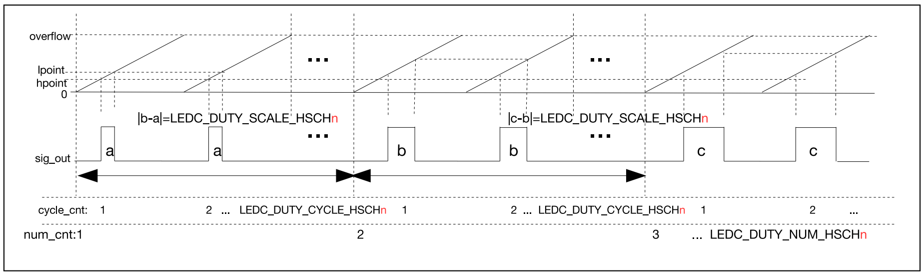


Figure 25: Output Signal Diagram of Gradient Duty Cycle

Figure 25 is an illustration of this. In this configuration, LEDC_DUTY_NUM_HSCH*n*_R increases by LEDC_DUTY_SCALE_HSCH*n* for every LEDC_DUTY_CYCLE_HSCH*n* clock cycles, which is reflected in the duty cycle of the output signal.

6.2.4 Interrupts

- LEDC_DUTY_CHNG_END_LSCH*n*_INT: Triggered when a fade on a low-speed channel has finished.
- LEDC_DUTY_CHNG_END_HSCH*n*_INT: Triggered when a fade on a high-speed channel has finished.
- LEDC_HS_TIMER*x*_OVF_INT: Triggered when a high-speed timer has reached its maximum counter value.
- LEDC_LS_TIMER*x*_OVF_INT: Triggered when a low-speed timer has reached its maximum counter value.

6.3 Register Summary

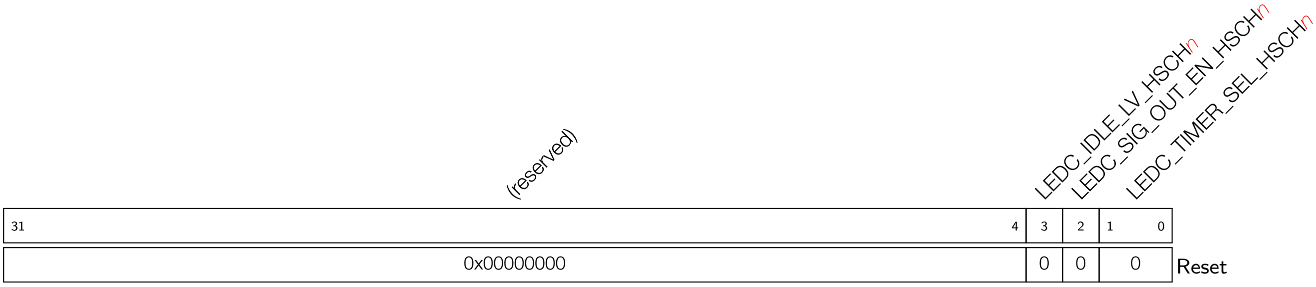
Name	Description	Address	Access
Configuration registers			
LEDC_CONF_REG	Global ledc configuration register	0x3FF59190	R/W
LEDC_HSCH0_CONF0_REG	Configuration register 0 for high-speed channel 0	0x3FF59000	R/W
LEDC_HSCH1_CONF0_REG	Configuration register 0 for high-speed channel 1	0x3FF59014	R/W
LEDC_HSCH2_CONF0_REG	Configuration register 0 for high-speed channel 2	0x3FF59028	R/W
LEDC_HSCH3_CONF0_REG	Configuration register 0 for high-speed channel 3	0x3FF5903C	R/W
LEDC_HSCH4_CONF0_REG	Configuration register 0 for high-speed channel 4	0x3FF59050	R/W

Name	Description	Address	Access
LEDC_HSCH5_CONF0_REG	Configuration register 0 for high-speed channel 5	0x3FF59064	R/W
LEDC_HSCH6_CONF0_REG	Configuration register 0 for high-speed channel 6	0x3FF59078	R/W
LEDC_HSCH7_CONF0_REG	Configuration register 0 for high-speed channel 7	0x3FF5908C	R/W
LEDC_HSCH0_CONF1_REG	Configuration register 1 for high-speed channel 0	0x3FF5900C	R/W
LEDC_HSCH1_CONF1_REG	Configuration register 1 for high-speed channel 1	0x3FF59020	R/W
LEDC_HSCH2_CONF1_REG	Configuration register 1 for high-speed channel 2	0x3FF59034	R/W
LEDC_HSCH3_CONF1_REG	Configuration register 1 for high-speed channel 3	0x3FF59048	R/W
LEDC_HSCH4_CONF1_REG	Configuration register 1 for high-speed channel 4	0x3FF5905C	R/W
LEDC_HSCH5_CONF1_REG	Configuration register 1 for high-speed channel 5	0x3FF59070	R/W
LEDC_HSCH6_CONF1_REG	Configuration register 1 for high-speed channel 6	0x3FF59084	R/W
LEDC_HSCH7_CONF1_REG	Configuration register 1 for high-speed channel 7	0x3FF59098	R/W
LEDC_LSCH0_CONF0_REG	Configuration register 0 for low-speed channel 0	0x3FF590A0	R/W
LEDC_LSCH1_CONF0_REG	Configuration register 0 for low-speed channel 1	0x3FF590B4	R/W
LEDC_LSCH2_CONF0_REG	Configuration register 0 for low-speed channel 2	0x3FF590C8	R/W
LEDC_LSCH3_CONF0_REG	Configuration register 0 for low-speed channel 3	0x3FF590DC	R/W
LEDC_LSCH4_CONF0_REG	Configuration register 0 for low-speed channel 4	0x3FF590F0	R/W
LEDC_LSCH5_CONF0_REG	Configuration register 0 for low-speed channel 5	0x3FF59104	R/W
LEDC_LSCH6_CONF0_REG	Configuration register 0 for low-speed channel 6	0x3FF59118	R/W
LEDC_LSCH7_CONF0_REG	Configuration register 0 for low-speed channel 7	0x3FF5912C	R/W
LEDC_LSCH0_CONF1_REG	Configuration register 1 for low-speed channel 0	0x3FF590AC	R/W
LEDC_LSCH1_CONF1_REG	Configuration register 1 for low-speed channel 1	0x3FF590C0	R/W
LEDC_LSCH2_CONF1_REG	Configuration register 1 for low-speed channel 2	0x3FF590D4	R/W
LEDC_LSCH3_CONF1_REG	Configuration register 1 for low-speed channel 3	0x3FF590E8	R/W
LEDC_LSCH4_CONF1_REG	Configuration register 1 for low-speed channel 4	0x3FF590FC	R/W
LEDC_LSCH5_CONF1_REG	Configuration register 1 for low-speed channel 5	0x3FF59110	R/W
LEDC_LSCH6_CONF1_REG	Configuration register 1 for low-speed channel 6	0x3FF59124	R/W
LEDC_LSCH7_CONF1_REG	Configuration register 1 for low-speed channel 7	0x3FF59138	R/W
Duty-cycle registers			
LEDC_HSCH0_DUTY_REG	Initial duty cycle for high-speed channel 0	0x3FF59008	R/W
LEDC_HSCH1_DUTY_REG	Initial duty cycle for high-speed channel 1	0x3FF5901C	R/W
LEDC_HSCH2_DUTY_REG	Initial duty cycle for high-speed channel 2	0x3FF59030	R/W
LEDC_HSCH3_DUTY_REG	Initial duty cycle for high-speed channel 3	0x3FF59044	R/W
LEDC_HSCH4_DUTY_REG	Initial duty cycle for high-speed channel 4	0x3FF59058	R/W
LEDC_HSCH5_DUTY_REG	Initial duty cycle for high-speed channel 5	0x3FF5906C	R/W
LEDC_HSCH6_DUTY_REG	Initial duty cycle for high-speed channel 6	0x3FF59080	R/W
LEDC_HSCH7_DUTY_REG	Initial duty cycle for high-speed channel 7	0x3FF59094	R/W
LEDC_HSCH0_DUTY_R_REG	Current duty cycle for high-speed channel 0	0x3FF59010	RO
LEDC_HSCH1_DUTY_R_REG	Current duty cycle for high-speed channel 1	0x3FF59024	RO
LEDC_HSCH2_DUTY_R_REG	Current duty cycle for high-speed channel 2	0x3FF59038	RO
LEDC_HSCH3_DUTY_R_REG	Current duty cycle for high-speed channel 3	0x3FF5904C	RO
LEDC_HSCH4_DUTY_R_REG	Current duty cycle for high-speed channel 4	0x3FF59060	RO
LEDC_HSCH5_DUTY_R_REG	Current duty cycle for high-speed channel 5	0x3FF59074	RO
LEDC_HSCH6_DUTY_R_REG	Current duty cycle for high-speed channel 6	0x3FF59088	RO
LEDC_HSCH7_DUTY_R_REG	Current duty cycle for high-speed channel 7	0x3FF5909C	RO

Name	Description	Address	Access
LEDC_LSCH0_DUTY_REG	Initial duty cycle for low-speed channel 0	0x3FF590A8	R/W
LEDC_LSCH1_DUTY_REG	Initial duty cycle for low-speed channel 1	0x3FF590BC	R/W
LEDC_LSCH2_DUTY_REG	Initial duty cycle for low-speed channel 2	0x3FF590D0	R/W
LEDC_LSCH3_DUTY_REG	Initial duty cycle for low-speed channel 3	0x3FF590E4	R/W
LEDC_LSCH4_DUTY_REG	Initial duty cycle for low-speed channel 4	0x3FF590F8	R/W
LEDC_LSCH5_DUTY_REG	Initial duty cycle for low-speed channel 5	0x3FF5910C	R/W
LEDC_LSCH6_DUTY_REG	Initial duty cycle for low-speed channel 6	0x3FF59120	R/W
LEDC_LSCH7_DUTY_REG	Initial duty cycle for low-speed channel 7	0x3FF59134	R/W
LEDC_LSCH0_DUTY_R_REG	Current duty cycle for low-speed channel 0	0x3FF590B0	RO
LEDC_LSCH1_DUTY_R_REG	Current duty cycle for low-speed channel 1	0x3FF590C4	RO
LEDC_LSCH2_DUTY_R_REG	Current duty cycle for low-speed channel 2	0x3FF590D8	RO
LEDC_LSCH3_DUTY_R_REG	Current duty cycle for low-speed channel 3	0x3FF590EC	RO
LEDC_LSCH4_DUTY_R_REG	Current duty cycle for low-speed channel 4	0x3FF59100	RO
LEDC_LSCH5_DUTY_R_REG	Current duty cycle for low-speed channel 5	0x3FF59114	RO
LEDC_LSCH6_DUTY_R_REG	Current duty cycle for low-speed channel 6	0x3FF59128	RO
LEDC_LSCH7_DUTY_R_REG	Current duty cycle for low-speed channel 7	0x3FF5913C	RO
Timer registers			
LEDC_HSTIMER0_CONF_REG	High-speed timer 0 configuration	0x3FF59140	R/W
LEDC_HSTIMER1_CONF_REG	High-speed timer 1 configuration	0x3FF59148	R/W
LEDC_HSTIMER2_CONF_REG	High-speed timer 2 configuration	0x3FF59150	R/W
LEDC_HSTIMER3_CONF_REG	High-speed timer 3 configuration	0x3FF59158	R/W
LEDC_HSTIMER0_VALUE_REG	High-speed timer 0 current counter value	0x3FF59144	RO
LEDC_HSTIMER1_VALUE_REG	High-speed timer 1 current counter value	0x3FF5914C	RO
LEDC_HSTIMER2_VALUE_REG	High-speed timer 2 current counter value	0x3FF59154	RO
LEDC_HSTIMER3_VALUE_REG	High-speed timer 3 current counter value	0x3FF5915C	RO
LEDC_LSTIMER0_CONF_REG	Low-speed timer 0 configuration	0x3FF59160	R/W
LEDC_LSTIMER1_CONF_REG	Low-speed timer 1 configuration	0x3FF59168	R/W
LEDC_LSTIMER2_CONF_REG	Low-speed timer 2 configuration	0x3FF59170	R/W
LEDC_LSTIMER3_CONF_REG	Low-speed timer 3 configuration	0x3FF59178	R/W
LEDC_LSTIMER0_VALUE_REG	Low-speed timer 0 current counter value	0x3FF59164	RO
LEDC_LSTIMER1_VALUE_REG	Low-speed timer 1 current counter value	0x3FF5916C	RO
LEDC_LSTIMER2_VALUE_REG	Low-speed timer 2 current counter value	0x3FF59174	RO
LEDC_LSTIMER3_VALUE_REG	Low-speed timer 3 current counter value	0x3FF5917C	RO
Interrupt registers			
LEDC_INT_RAW_REG	Raw interrupt status	0x3FF59180	RO
LEDC_INT_ST_REG	Masked interrupt status	0x3FF59184	RO
LEDC_INT_ENA_REG	Interrupt enable bits	0x3FF59188	R/W
LEDC_INT_CLR_REG	Interrupt clear bits	0x3FF5918C	WO

6.4 Registers

Register 6.1: LEDC_HSCHn_CONF0_REG (n: 0-7) (0x1C+0x10*n)



LEDC_IDLE_LV_HSCHn This bit is used to control the output value when high-speed channel n is inactive. (R/W)

LEDC_SIG_OUT_EN_HSCHn This is the output enable control bit for high-speed channel n. (R/W)

LEDC_TIMER_SEL_HSCHn There are four high-speed timers. These two bits are used to select one of them for high-speed channel n: (R/W)

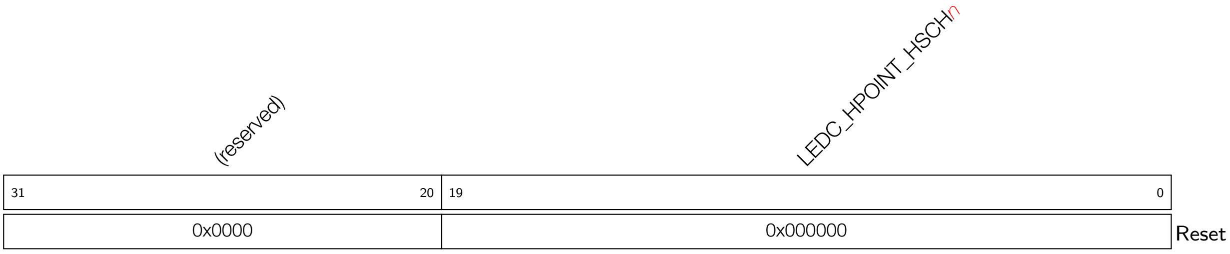
0: select hstimer0;

1: select hstimer1;

2: select hstimer2;

3: select hstimer3.

Register 6.2: LEDC_HSCHn_HPOINT_REG (n: 0-7) (0x20+0x10*n)



LEDC_HPOINT_HSCHn The output value changes to high when htimerx(x=[0,3]), selected by high-speed channel n, has reached reg_hpoint_hschn[19:0]. (R/W)

Register 6.3: LEDC_HSCHn_DUTY_REG (n: 0-7) (0x24+0x10*n)

(reserved)	LEDC_DUTY_HSCHn	
31 25	24 0	
0x00	0x0000000	Reset

LEDC_DUTY_HSCHn The register is used to control output duty. When hstimerx(x=[0,3]), selected by high-speed channel n, has reached reg_lpoint_hschn, the output signal changes to low. (R/W)

reg_lpoint_hschn=(reg_hpoint_hschn[19:0]+reg_duty_hschn[24:4]) (1)

reg_lpoint_hschn=(reg_hpoint_hschn[19:0]+reg_duty_hschn[24:4] +1) (2)

See the Functional Description for more information on when (1) or (2) is chosen.

Register 6.4: LEDC_HSCHn_CONF1_REG (n: 0-7) (0x28+0x10*n)

LEDC_DUTY_START_HSCHn	LEDC_DUTY_INC_HSCHn	LEDC_DUTY_NUM_HSCHn	LEDC_DUTY_CYCLE_HSCHn	LEDC_DUTY_SCALE_HSCHn	
31	30	29 20	19 10	9 0	
0	1	0x000	0x000	0x000	Reset

LEDC_DUTY_START_HSCHn When REG_DUTY_NUM_HSCHn, REG_DUTY_CYCLE_HSCHn and REG_DUTY_SCALE_HSCHn has been configured, these register will not take effect until REG_DUTY_START_HSCHn is set. This bit is automatically cleared by hardware. (R/W)

LEDC_DUTY_INC_HSCHn This register is used to increase or decrease the duty of output signal for high-speed channel n. (R/W)

LEDC_DUTY_NUM_HSCHn This register is used to control the number of times the duty cycle is increased or decreased for high-speed channel n. (R/W)

LEDC_DUTY_CYCLE_HSCHn This register is used to increase or decrease the duty cycle every time REG_DUTY_CYCLE_HSCHn cycles for high-speed channel n. (R/W)

LEDC_DUTY_SCALE_HSCHn This register is used to increase or decrease the step scale for high-speed channel n. (R/W)

Register 6.5: LEDC_HSCH*n*_DUTY_R_REG (*n*: 0-7) (0x2C+0x10**n*)

(reserved)	LEDC_DUTY_HSCH*n*_R	
31 25	24 0	
0x00	0x0000000	Reset

LEDC_DUTY_HSCH*n*_R This register represents the current duty cycle of the output signal for high-speed channel *n*. (RO)

Register 6.6: LEDC_LSCH*n*_CONF0_REG (*n*: 0-7) (0xBC+0x10**n*)

(reserved)	LEDC_PARA_UP_LSCH*n*	LEDC_IDLE_LV_LSCH*n*	LEDC_SIG_OUT_EN_LSCH*n*	LEDC_TIMER_SEL_LSCH*n*	
31 5	4	3	2	1 0	
0x0000000	0	0	0	0	Reset

LEDC_PARA_UP_LSCH*n* This bit is used to update register LEDC_LSCH*n*_HPOINT and LEDC_LSCH*n*_DUTY for low-speed channel *n*. (R/W)

LEDC_IDLE_LV_LSCH*n* This bit is used to control the output value, when low-speed channel *n* is inactive. (R/W)

LEDC_SIG_OUT_EN_LSCH*n* This is the output enable control bit for low-speed channel *n*. (R/W)

LEDC_TIMER_SEL_LSCH*n* There are four low-speed timers, the two bits are used to select one of them for low-speed channel *n*. (R/W)

0: select lstimer0;

1: select lstimer1;

2: select lstimer2;

3: select lstimer3.

Register 6.7: LEDC_LSCHn_HPOINT_REG (n: 0-7) (0xC0+0x10*n)

(reserved)	LEDC_HPOINT_LSCHn	
31 20	19 0	
0x0000	0x000000	Reset

LEDC_HPOINT_LSCHn The output value changes to high when lstimerx(x=[0,3]), selected by low-speed channel n, has reached reg_hpoint_lschn[19:0]. (R/W)

Register 6.8: LEDC_LSCHn_DUTY_REG (n: 0-7) (0xC4+0x10*n)

(reserved)	LEDC_DUTY_LSCHn	
31 25	24 0	
0x00	0x0000000	Reset

LEDC_DUTY_LSCHn The register is used to control output duty. When lstimerx(x=[0,3]), chosen by low-speed channel n, has reached reg_lpoint_lschn,the output signal changes to low. (R/W)

reg_lpoint_lschn=(reg_hpoint_lschn[19:0]+reg_duty_lschn[24:4]) (1)

reg_lpoint_lschn=(reg_hpoint_lschn[19:0]+reg_duty_lschn[24:4] +1) (2)

See the Functional Description for more information on when (1) or (2) is chosen.

Register 6.9: LEDC_LSCH*n*_CONF1_REG (*n*: 0-7) (0xC8+0x10**n*)

LEDC_DUTY_START_LSCH*n*	LEDC_DUTY_INC_LSCH*n*	LEDC_DUTY_NUM_LSCH*n*	LEDC_DUTY_CYCLE_LSCH*n*	LEDC_DUTY_SCALE_LSCH*n*	
31	30	29 20	19 10	9 0	
0	1	0x000	0x000	0x000	Reset

LEDC_DUTY_START_LSCH*n* When reg_duty_num_hsch*n*, reg_duty_cycle_hsch*n* and reg_duty_scale_hsch*n* have been configured, these settings will not take effect until set reg_duty_start_hsch*n*. This bit is automatically cleared by hardware. (R/W)

LEDC_DUTY_INC_LSCH*n* This register is used to increase or decrease the duty of output signal for low-speed channel *n*. (R/W)

LEDC_DUTY_NUM_LSCH*n* This register is used to control the number of times the duty cycle is increased or decreased for low-speed channel *n*. (R/W)

LEDC_DUTY_CYCLE_LSCH*n* This register is used to increase or decrease the duty every reg_duty_cycle_lsch*n* cycles for low-speed channel *n*. (R/W)

LEDC_DUTY_SCALE_LSCH*n* This register is used to increase or decrease the step scale for low-speed channel *n*. (R/W)

Register 6.10: LEDC_LSCH*n*_DUTY_R_REG (*n*: 0-7) (0xCC+0x10**n*)

(reserved)	LEDC_DUTY_LSCH*n*_R	
31 25	24 0	
0x00	0x0000000	Reset

LEDC_DUTY_LSCH*n*_R This register represents the current duty of the output signal for low-speed channel *n*. (RO)

Register 6.11: LEDC_HSTIMERx_CONF_REG (x: 0-3) (0x140+8*x)

(reserved)	LEDC_TICK_SEL_HSTIMERx	LEDC_HSTIMERx_RST	LEDC_HSTIMERx_PAUSE	LEDC_DIV_NUM_HSTIMERx	LEDC_HSTIMERx_LIM	
31 26	25	24	23	22 5	4 0	
0x00	0	1	0	0x00000	0x00	Reset

LEDC_TICK_SEL_HSTIMERx This bit is used to select APB_CLK or REF_TICK for high-speed timer x. (R/W)

1: APB_CLK;

0: REF_TICK.

LEDC_HSTIMERx_RST This bit is used to reset high-speed timer x. The counter value will be 'zero' after reset. (R/W)

LEDC_HSTIMERx_PAUSE This bit is used to suspend the counter in high-speed timer x. (R/W)

LEDC_DIV_NUM_HSTIMERx This register is used to configure the division factor for the divider in high-speed timer x. The least significant eight bits represent the fractional part. (R/W)

LEDC_HSTIMERx_LIM This register is used to control the range of the counter in high-speed timer x. The counter range is [0,2**reg_hstimerx_lim], the maximum bit width for counter is 20. (R/W)

Register 6.12: LEDC_HSTIMERx_VALUE_REG (x: 0-3) (0x144+8*x)

(reserved)	LEDC_HSTIMERx_CNT	
31 20	19 0	
0x0000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

LEDC_HSTIMERx_CNT Software can read this register to get the current counter value of high-speed timer x. (RO)

Register 6.13: LEDC_LSTIMERx_CONF_REG (x: 0-3) (0x160+8*x)

(reserved)	LEDC_LSTIMERx_PARA_UP	LEDC_TICK_SEL_LSTIMERx	LEDC_LSTIMERx_RST	LEDC_LSTIMERx_PAUSE	LEDC_DIV_NUM_LSTIMERx	LEDC_LSTIMERx_LIM	
31 27	26	25	24	23	22 5	4 0	
0x00	0	0	1	0	0x00000	0x00	Reset

LEDC_LSTIMERx_PARA_UP Set this bit to update REG_DIV_NUM_LSTIMEx and REG_LSTIMERx_LIM. (R/W)

LEDC_TICK_SEL_LSTIMERx This bit is used to select SLOW_CLK or REF_TICK for low-speed timer x. (R/W)

1: SLOW_CLK;

0: REF_TICK.

LEDC_LSTIMERx_RST This bit is used to reset low-speed timer x. The counter will show 0 after reset. (R/W)

LEDC_LSTIMERx_PAUSE This bit is used to suspend the counter in low-speed timer x. (R/W)

LEDC_DIV_NUM_LSTIMERx This register is used to configure the division factor for the divider in low-speed timer x. The least significant eight bits represent the fractional part. (R/W)

LEDC_LSTIMERx_LIM This register is used to control the range of the counter in low-speed timer x. The counter range is [0,2**reg_lstimerx_lim], the max bit width for counter is 20. (R/W)

Register 6.14: LEDC_LSTIMERx_VALUE_REG (x: 0-3) (0x164+8*x)

(reserved)	LEDC_LSTIMERx_CNT	
31 20	19 0	
0x0000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

LEDC_LSTIMERx_CNT Software can read this register to get the current counter value of low-speed timer x. (RO)

Register 6.15: LEDC_INT_RAW_REG (0x0180)

(reserved)	LEDC_DUTY_CHNG_END_LSCH7_INT_RAW	LEDC_DUTY_CHNG_END_LSCH6_INT_RAW	LEDC_DUTY_CHNG_END_LSCH5_INT_RAW	LEDC_DUTY_CHNG_END_LSCH4_INT_RAW	LEDC_DUTY_CHNG_END_LSCH3_INT_RAW	LEDC_DUTY_CHNG_END_LSCH2_INT_RAW	LEDC_DUTY_CHNG_END_LSCH1_INT_RAW	LEDC_DUTY_CHNG_END_LSCH0_INT_RAW	LEDC_DUTY_CHNG_END_HSCH7_INT_RAW	LEDC_DUTY_CHNG_END_HSCH6_INT_RAW	LEDC_DUTY_CHNG_END_HSCH5_INT_RAW	LEDC_DUTY_CHNG_END_HSCH4_INT_RAW	LEDC_DUTY_CHNG_END_HSCH3_INT_RAW	LEDC_DUTY_CHNG_END_HSCH2_INT_RAW	LEDC_DUTY_CHNG_END_HSCH1_INT_RAW	LEDC_DUTY_CHNG_END_HSCH0_INT_RAW	LEDC_LSTIMER3_OVF_INT_RAW	LEDC_LSTIMER2_OVF_INT_RAW	LEDC_LSTIMER1_OVF_INT_RAW	LEDC_LSTIMER0_OVF_INT_RAW	LEDC_HSTIMER3_OVF_INT_RAW	LEDC_HSTIMER2_OVF_INT_RAW	LEDC_HSTIMER1_OVF_INT_RAW	LEDC_HSTIMER0_OVF_INT_RAW	
31 24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_DUTY_CHNG_END_LSCH*n*_INT_RAW The raw interrupt status bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (RO)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_RAW The raw interrupt status bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (RO)

LEDC_LSTIMER*x*_OVF_INT_RAW The raw interrupt status bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (RO)

LEDC_HSTIMER*x*_OVF_INT_RAW The raw interrupt status bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (RO)

Register 6.16: LEDC_INT_ST_REG (0x0184)

(reserved)	LEDC_DUTY_CHNG_END_LSCH7_INT_ST	LEDC_DUTY_CHNG_END_LSCH6_INT_ST	LEDC_DUTY_CHNG_END_LSCH5_INT_ST	LEDC_DUTY_CHNG_END_LSCH4_INT_ST	LEDC_DUTY_CHNG_END_LSCH3_INT_ST	LEDC_DUTY_CHNG_END_LSCH2_INT_ST	LEDC_DUTY_CHNG_END_LSCH1_INT_ST	LEDC_DUTY_CHNG_END_LSCH0_INT_ST	LEDC_DUTY_CHNG_END_HSCH7_INT_ST	LEDC_DUTY_CHNG_END_HSCH6_INT_ST	LEDC_DUTY_CHNG_END_HSCH5_INT_ST	LEDC_DUTY_CHNG_END_HSCH4_INT_ST	LEDC_DUTY_CHNG_END_HSCH3_INT_ST	LEDC_DUTY_CHNG_END_HSCH2_INT_ST	LEDC_DUTY_CHNG_END_HSCH1_INT_ST	LEDC_DUTY_CHNG_END_HSCH0_INT_ST	LEDC_LSTIMER3_OVF_INT_ST	LEDC_LSTIMER2_OVF_INT_ST	LEDC_LSTIMER1_OVF_INT_ST	LEDC_LSTIMER0_OVF_INT_ST	LEDC_HSTIMER3_OVF_INT_ST	LEDC_HSTIMER2_OVF_INT_ST	LEDC_HSTIMER1_OVF_INT_ST	LEDC_HSTIMER0_OVF_INT_ST	
31 24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_DUTY_CHNG_END_LSCH*n*_INT_ST The masked interrupt status bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (RO)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_ST The masked interrupt status bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (RO)

LEDC_LSTIMER*x*_OVF_INT_ST The masked interrupt status bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (RO)

LEDC_HSTIMER*x*_OVF_INT_ST The masked interrupt status bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (RO)

Register 6.17: LEDC_INT_ENA_REG (0x0188)

(reserved)	LEDC_DUTY_CHNG_END_LSCH7_INT_ENA	LEDC_DUTY_CHNG_END_LSCH6_INT_ENA	LEDC_DUTY_CHNG_END_LSCH5_INT_ENA	LEDC_DUTY_CHNG_END_LSCH4_INT_ENA	LEDC_DUTY_CHNG_END_LSCH3_INT_ENA	LEDC_DUTY_CHNG_END_LSCH2_INT_ENA	LEDC_DUTY_CHNG_END_LSCH1_INT_ENA	LEDC_DUTY_CHNG_END_LSCH0_INT_ENA	LEDC_DUTY_CHNG_END_HSCH7_INT_ENA	LEDC_DUTY_CHNG_END_HSCH6_INT_ENA	LEDC_DUTY_CHNG_END_HSCH5_INT_ENA	LEDC_DUTY_CHNG_END_HSCH4_INT_ENA	LEDC_DUTY_CHNG_END_HSCH3_INT_ENA	LEDC_DUTY_CHNG_END_HSCH2_INT_ENA	LEDC_DUTY_CHNG_END_HSCH1_INT_ENA	LEDC_DUTY_CHNG_END_HSCH0_INT_ENA	LEDC_LSTIMER3_OVF_INT_ENA	LEDC_LSTIMER2_OVF_INT_ENA	LEDC_LSTIMER1_OVF_INT_ENA	LEDC_LSTIMER0_OVF_INT_ENA	LEDC_HSTIMER3_OVF_INT_ENA	LEDC_HSTIMER2_OVF_INT_ENA	LEDC_HSTIMER1_OVF_INT_ENA	LEDC_HSTIMER0_OVF_INT_ENA	
31 24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_DUTY_CHNG_END_LSCH*n*_INT_ENA The interrupt enable bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (R/W)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_ENA The interrupt enable bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (R/W)

LEDC_LSTIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (R/W)

LEDC_HSTIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (R/W)

Register 6.18: LEDC_INT_CLR_REG (0x018C)

(reserved)	LEDC_DUTY_CHNG_END_LSCH7_INT_CLR	LEDC_DUTY_CHNG_END_LSCH6_INT_CLR	LEDC_DUTY_CHNG_END_LSCH5_INT_CLR	LEDC_DUTY_CHNG_END_LSCH4_INT_CLR	LEDC_DUTY_CHNG_END_LSCH3_INT_CLR	LEDC_DUTY_CHNG_END_LSCH2_INT_CLR	LEDC_DUTY_CHNG_END_LSCH1_INT_CLR	LEDC_DUTY_CHNG_END_LSCH0_INT_CLR	LEDC_DUTY_CHNG_END_HSCH7_INT_CLR	LEDC_DUTY_CHNG_END_HSCH6_INT_CLR	LEDC_DUTY_CHNG_END_HSCH5_INT_CLR	LEDC_DUTY_CHNG_END_HSCH4_INT_CLR	LEDC_DUTY_CHNG_END_HSCH3_INT_CLR	LEDC_DUTY_CHNG_END_HSCH2_INT_CLR	LEDC_DUTY_CHNG_END_HSCH1_INT_CLR	LEDC_DUTY_CHNG_END_HSCH0_INT_CLR	LEDC_LSTIMER3_OVF_INT_CLR	LEDC_LSTIMER2_OVF_INT_CLR	LEDC_LSTIMER1_OVF_INT_CLR	LEDC_LSTIMER0_OVF_INT_CLR	LEDC_HSTIMER3_OVF_INT_CLR	LEDC_HSTIMER2_OVF_INT_CLR	LEDC_HSTIMER1_OVF_INT_CLR	LEDC_HSTIMER0_OVF_INT_CLR	
31 24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_DUTY_CHNG_END_LSCH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (WO)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (WO)

LEDC_LSTIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_LSTIMER*x*_OVF_INT interrupt. (WO)

LEDC_HSTIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_HSTIMER*x*_OVF_INT interrupt. (WO)

Register 6.19: LEDC_CONF_REG (0x0190)

(reserved)	LEDC_APB_CLK_SEL	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

LEDC_APB_CLK_SEL This bit is used to set the frequency of SLOW_CLK. (R/W)

0: 8 MHz;

1: 80 MHz.

7. Remote Controller Peripheral

7.1 Introduction

The RMT (Remote Control) module is primarily designed to send and receive infrared remote control signals that use on-off-keying of a carrier frequency, but due to its design it can be used to generate various types of signals. An RMT transmitter does this by reading consecutive duration values for an active and inactive output from the built-in RAM block, optionally modulating it with a carrier wave. A receiver will inspect its input signal, optionally filtering it, and will place the lengths of time the signal is active and inactive in the RAM block.

The RMT module has eight channels, numbered zero to seven; registers, signals and blocks that are duplicated in each channel are indicated by an *n* which is used as a placeholder for the channel number.

7.2 Functional Description

7.2.1 RMT Architecture

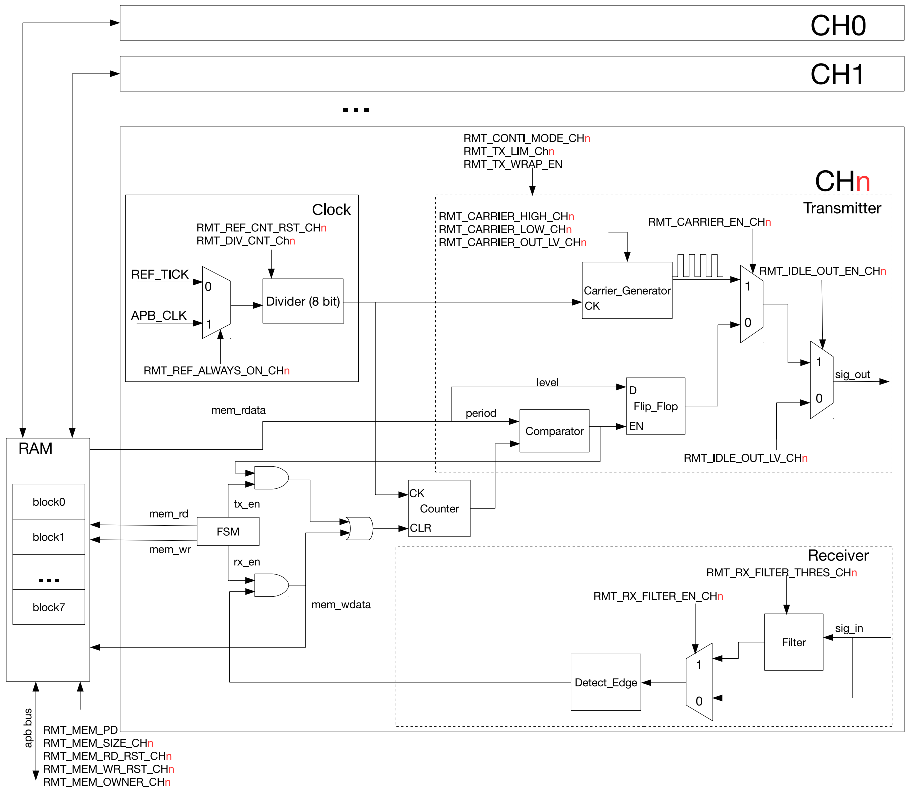


Figure 26: RMT Architecture

The RMT module contains eight channels; each channel has a transmitter and receiver, of which one can be active per channel. The eight channels share a 512x32-bit RAM block which can be read and written by the processor cores over the APB bus, read by the transmitters, and written by the receivers. The transmitted signal can optionally be modulated by a carrier wave. Each channel is clocked by a divided-down signal derived from either the APB bus clock or REF_TICK.

7.2.2 RMT RAM

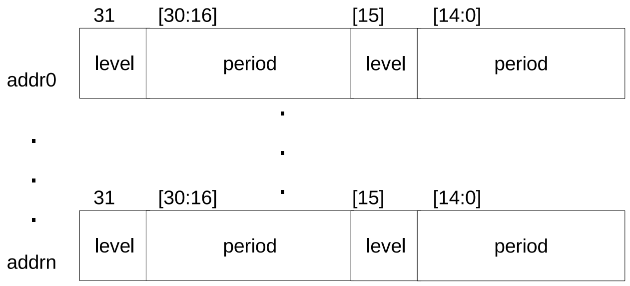


Figure 27: Data Structure

The data structure in RAM is shown in Figure 27. Each 32-bit value contains two 16-bit entries, containing two fields each: "level" indicates whether a high-/low-level value was received or is going to be sent, and "period" is the duration (in channel clock periods) for which the level lasts. A zero period is interpreted as an end-marker: the transmitter will stop transmitting once it has read this, and the receiver will write this, once it has detected that the signal it received has gone idle.

Normally, only one block of 64x32-bit worth of data can be sent or received. If the data size is larger than this block size, blocks can either be extended or the channel can be configured for wraparound mode.

The RMT RAM can be accessed via APB bus. The initial address is RMT base address + 0x800. The RAM block is divided into eight 64x32-bit blocks. By default, each channel uses one block (block zero for channel zero, block one for channel one, and so on). Users can extend the memory for a specific channel by configuring RMT_MEM_SIZE_CH*n* register; setting this to >1 will prompt the channel to use the memory of subsequent channels as well. The RAM address range for channel *n* is start_addr_CH*n* to end_addr_CH*n*, which are defined by:

start_addr_ch*n* $=$ RMT base address $+$ 0x800 $+ 64 * 4 * n$, and

end_addr_ch*n* $=$ RMT base address $+$ 0x800 $+ \; 64 * 4 * n + 64 * 4 *$ RMT_MEM_SIZE_CH*n* $\bmod 512 * 4$

To protect a receiver from overwriting the blocks a transmitter is about to transmit, RMT_MEM_OWNER_CH*n* can be configured to assign the owner, i.e. transmitter or receiver, of channel *n*'s RAM block. If this ownership is violated, the RMT_CH*n*_ERR interrupt will be generated.

7.2.3 Clock

The main clock for a channel is generated by taking either the 80 MHz APB clock or REF_TICK (usually 1MHz), according to the state of RMT_REF_ALWAYS_ON_CH*n*. (For more information on the clock sources, please see Chapter Reset And Clock.) Then, the aforementioned state gets scaled down using a configurable 8-bit divider to create the channel clock which is used by both the carrier wave generator and the counter. The divider value can be set by configuring RMT_DIV_CNT_CH*n*.

7.2.4 Transmitter

When the RMT_TX_START_CH*n* register is 1, the transmitter of channel *n* will start reading data from RAM and sending it. The transmitter will receive a 32-bits value each time it reads from RAM. Of these 32 bits, the low 16-bit entry is sent first and the high entry second.

To transmit more data than can be fitted in the channel's RAM, wraparound mode can be enabled. In this mode, when the transmitter has reached the last entry in the channel's memory, it will loop back to the first byte. To use this mechanism to send more data than can be fitted in the channel's RAM, fill the RAM with the initial events and set RMT_CHn_TX_LIM_REG to cause an RMT_CHn_TX_THR_EVENT_INT interrupt before the wraparound happens. Then, when the interrupt happens, the already sent data should be replaced by subsequent events: when the wraparound happens the transmitter will seamlessly continue sending the new events.

With or without wraparound mode enabled, transmission ends when an entry with zero length is encountered. When this happens, the transmitter will generate a RMT_CHn_TX_END_INT interrupt, and return to the idle state. When a transmitter is in the idle state, users can configure RMT_IDLE_OUT_EN_CHn and RMT_IDLE_OUT_LV_CHn to control the transmitter output manually.

The output of the transmitter can be modulated using a carrier wave by setting RMT_CARRIER_EN_CHn. The carrier frequency and duty cycle can be configured by adjusting its high and low durations in channel clock cycles, in RMT_CARRIER_HIGH_CHn and RMT_CARRIER_HIGH_CHn.

7.2.5 Receiver

When RMT_RX_EN_CHn is set to 1, the receiver in channel n becomes active, measuring the duration between input signal edges. These will be written as period/level value pairs to the channel RAM in the same fashion as the transmitter sends them. Receiving ends when the receiver detects no change in signal level for more than RMT_IDLE_THRES_CHn channel clock ticks; the receiver will write a final entry with 0 period, generate an RMT_CHn_RX_END_INT_RAW interrupt, and return to the idle state.

The receiver has an input signal filter which can be configured using RMT_RX_FILTER_EN_CHn: The filter will remove pulses with a length of less than RMT_RX_FILTER_THRES_CHn in APB clock periods.

When the RMT module is inactive, the RAM can be put into low-power mode by setting the RMT_MEM_PD register to 1.

7.2.6 Interrupts

- RMT_CHn_TX_THR_EVENT_INT: Triggered when the number of events the transmitter has sent matches the contents of the RMT_CHn_TX_LIM_REG register.
- RMT_CHn_TX_END_INT: Triggered when the transmitter has finished transmitting the signal.
- RMT_CHn_RX_END_INT: Triggered when the receiver has finished receiving a signal.

7.3 Register Summary

Name	Description	Address	Access
Configuration registers			
RMT_CH0CONF0_REG	Channel 0 config register 0	0x3FF56020	R/W
RMT_CH0CONF1_REG	Channel 0 config register 1	0x3FF56024	R/W
RMT_CH1CONF0_REG	Channel 1 config register 0	0x3FF56028	R/W
RMT_CH1CONF1_REG	Channel 1 config register 1	0x3FF5602C	R/W
RMT_CH2CONF0_REG	Channel 2 config register 0	0x3FF56030	R/W
RMT_CH2CONF1_REG	Channel 2 config register 1	0x3FF56034	R/W
RMT_CH3CONF0_REG	Channel 3 config register 0	0x3FF56038	R/W
RMT_CH3CONF1_REG	Channel 3 config register 1	0x3FF5603C	R/W
RMT_CH4CONF0_REG	Channel 4 config register 0	0x3FF56040	R/W

RMT_CH4CONF1_REG	Channel 4 config register 1	0x3FF56044	R/W
RMT_CH5CONF0_REG	Channel 5 config register 0	0x3FF56048	R/W
RMT_CH5CONF1_REG	Channel 5 config register 1	0x3FF5604C	R/W
RMT_CH6CONF0_REG	Channel 6 config register 0	0x3FF56050	R/W
RMT_CH6CONF1_REG	Channel 6 config register 1	0x3FF56054	R/W
RMT_CH7CONF0_REG	Channel 7 config register 0	0x3FF56058	R/W
RMT_CH7CONF1_REG	Channel 7 config register 1	0x3FF5605C	R/W
Interrupt registers			
RMT_INT_RAW_REG	Raw interrupt status	0x3FF560A0	RO
RMT_INT_ST_REG	Masked interrupt status	0x3FF560A4	RO
RMT_INT_ENA_REG	Interrupt enable bits	0x3FF560A8	R/W
RMT_INT_CLR_REG	Interrupt clear bits	0x3FF560AC	WO
Carrier wave duty cycle registers			
RMT_CH0CARRIER_DUTY_REG	Channel 0 duty cycle configuration register	0x3FF560B0	R/W
RMT_CH1CARRIER_DUTY_REG	Channel 1 duty cycle configuration register	0x3FF560B4	R/W
RMT_CH2CARRIER_DUTY_REG	Channel 2 duty cycle configuration register	0x3FF560B8	R/W
RMT_CH3CARRIER_DUTY_REG	Channel 3 duty cycle configuration register	0x3FF560BC	R/W
RMT_CH4CARRIER_DUTY_REG	Channel 4 duty cycle configuration register	0x3FF560C0	R/W
RMT_CH5CARRIER_DUTY_REG	Channel 5 duty cycle configuration register	0x3FF560C4	R/W
RMT_CH6CARRIER_DUTY_REG	Channel 6 duty cycle configuration register	0x3FF560C8	R/W
RMT_CH7CARRIER_DUTY_REG	Channel 7 duty cycle configuration register	0x3FF560CC	R/W
Tx event configuration registers			
RMT_CH0_TX_LIM_REG	Channel 0 Tx event configuration register	0x3FF560D0	R/W
RMT_CH1_TX_LIM_REG	Channel 1 Tx event configuration register	0x3FF560D4	R/W
RMT_CH2_TX_LIM_REG	Channel 2 Tx event configuration register	0x3FF560D8	R/W
RMT_CH3_TX_LIM_REG	Channel 3 Tx event configuration register	0x3FF560DC	R/W
RMT_CH4_TX_LIM_REG	Channel 4 Tx event configuration register	0x3FF560E0	R/W
RMT_CH5_TX_LIM_REG	Channel 5 Tx event configuration register	0x3FF560E4	R/W
RMT_CH6_TX_LIM_REG	Channel 6 Tx event configuration register	0x3FF560E8	R/W
RMT_CH7_TX_LIM_REG	Channel 7 Tx event configuration register	0x3FF560EC	R/W
Other registers			
RMT_APB_CONF_REG	RMT-wide configuration register	0x3FF560F0	R/W

7.4 Registers

Register 7.1: RMT_CH*n*CONF0_REG (*n*: 0-7) (0x0058+8n*)**

(reserved)	RMT_MEM_PD	RMT_CARRIER_OUT_LV_CH*n*	RMT_CARRIER_EN_CH*n*	RMT_MEM_SIZE_CH*n*	RMT_IDLE_THRES_CH*n*	RMT_DIV_CNT_CH*n*	
31	30	29	28	27 24	23 8	7 0	
0x0	0	1	1	0x01	0x01000	0x002	Reset

RMT_MEM_PD This bit is used to power down the entire RMT RAM block. (It only exists in RMT_CH0CONF0). 1: power down memory; 0: power up memory. (R/W)

RMT_CARRIER_OUT_LV_CH*n* This bit is used for configuration when the carrier wave is being transmitted. Transmit on low output level with 1, and transmit on high output level with 0. (R/W)

RMT_CARRIER_EN_CH*n* This is the carrier modulation enable control bit for channel*n*. Carrier modulation is enabled with 1, while carrier modulation is disabled with 0. (R/W)

RMT_MEM_SIZE_CH*n* This register is used to configure the amount of memory blocks allocated to channel *n* (R/W)

RMT_IDLE_THRES_CH*n* In receive mode, when no edge is detected on the input signal for longer than reg_idle_thres_ch*n* channel clock cycles, the receive process is finished. (R/W)

RMT_DIV_CNT_CH*n* This register is used to set the divider for the channel clock of channel *n*. (R/W)

Register 7.2: RMT_CH*n*CONF1_REG (*n*: 0-7) (0x005c+8**n*)

	(reserved)	RMT_IDLE_OUT_EN_CH*n*	RMT_IDLE_OUT_LV_CH*n*	RMT_REF_ALWAYS_ON_CH*n*	RMT_REF_CNT_RST_CH*n*	RMT_RX_FILTER_THRES_CH*n*	RMT_RX_FILTER_EN_CH*n*	RMT_TX_CONTI_MODE_CH*n*	RMT_MEM_OWNER_CH*n*	(reserved)	RMT_MEM_RD_RST_CH*n*	RMT_MEM_WR_RST_CH*n*	RMT_RX_EN_CH*n*	RMT_TX_START_CH*n*
	31 20	19	18	17	16	15 8	7	6	5	4	3	2	1	0
Reset	0x0000	0	0	0	0	0x00F	0	0	1	0	0	0	0	0

RMT_IDLE_OUT_EN_CH*n* This is the output enable control bit for channel *n* in IDLE state. (R/W)

RMT_IDLE_OUT_LV_CH*n* This bit configures the output signals level for channel *n* in IDLE state. (R/W)

RMT_REF_ALWAYS_ON_CH*n* This bit is used to select the channel's base clock. 1:clk_apb; 0:clk_ref. (R/W)

RMT_REF_CNT_RST_CH*n* Setting this bit resets the clock divider of channel *n*. (R/W)

RMT_RX_FILTER_THRES_CH*n* In receive mode, channel *n* ignores input pulse when the pulse width is smaller than this value in APB clock periods. (R/W)

RMT_RX_FILTER_EN_CH*n* This is the receive filter enable bit for channel *n*. (R/W)

RMT_TX_CONTI_MODE_CH*n* If this bit is set, instead of going to idle when the transmission ends, the transmitter will restart transmission. This results in a repeating output signal. (R/W)

RMT_MEM_OWNER_CH*n* This bit marks channel *n*'s RAM block ownership. Number 1 stands for the receiver using the RAM, while 0 stands for the transmitter using the RAM. (R/W)

RMT_MEM_RD_RST_CH*n* Set this bit to reset read RAM address for channel *n* by transmitter access. (R/W)

RMT_MEM_WR_RST_CH*n* Set this bit to reset write RAM address for channel *n* by receiver access. (R/W)

RMT_RX_EN_CH*n* Set this bit to enable receiving data on channel *n*. (R/W)

RMT_TX_START_CH*n* Set this bit to start sending data on channel *n*. (R/W)

Register 7.3: RMT_INT_RAW_REG (0x00a0)

RMT_CH7_TX_THR_EVENT_INT_RAW	RMT_CH6_TX_THR_EVENT_INT_RAW	RMT_CH5_TX_THR_EVENT_INT_RAW	RMT_CH4_TX_THR_EVENT_INT_RAW	RMT_CH3_TX_THR_EVENT_INT_RAW	RMT_CH2_TX_THR_EVENT_INT_RAW	RMT_CH1_TX_THR_EVENT_INT_RAW	RMT_CH0_TX_THR_EVENT_INT_RAW	RMT_CH7_ERR_INT_RAW	RMT_CH7_RX_END_INT_RAW	RMT_CH7_TX_END_INT_RAW	RMT_CH6_ERR_INT_RAW	RMT_CH6_RX_END_INT_RAW	RMT_CH6_TX_END_INT_RAW	RMT_CH5_ERR_INT_RAW	RMT_CH5_RX_END_INT_RAW	RMT_CH5_TX_END_INT_RAW	RMT_CH4_ERR_INT_RAW	RMT_CH4_RX_END_INT_RAW	RMT_CH4_TX_END_INT_RAW	RMT_CH3_ERR_INT_RAW	RMT_CH3_RX_END_INT_RAW	RMT_CH3_TX_END_INT_RAW	RMT_CH2_ERR_INT_RAW	RMT_CH2_RX_END_INT_RAW	RMT_CH2_TX_END_INT_RAW	RMT_CH1_ERR_INT_RAW	RMT_CH1_RX_END_INT_RAW	RMT_CH1_TX_END_INT_RAW	RMT_CH0_ERR_INT_RAW	RMT_CH0_RX_END_INT_RAW	RMT_CH0_TX_END_INT_RAW	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_THR_EVENT_INT_RAW The raw interrupt status bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (RO)

RMT_CH*n*_ERR_INT_RAW The raw interrupt status bit for the RMT_CH*n*_ERR_INT interrupt. (RO)

RMT_CH*n*_RX_END_INT_RAW The raw interrupt status bit for the RMT_CH*n*_RX_END_INT interrupt. (RO)

RMT_CH*n*_TX_END_INT_RAW The raw interrupt status bit for the RMT_CH*n*_TX_END_INT interrupt. (RO)

Register 7.4: RMT_INT_ST_REG (0x00a4)

RMT_CH7_TX_THR_EVENT_INT_ST	RMT_CH6_TX_THR_EVENT_INT_ST	RMT_CH5_TX_THR_EVENT_INT_ST	RMT_CH4_TX_THR_EVENT_INT_ST	RMT_CH3_TX_THR_EVENT_INT_ST	RMT_CH2_TX_THR_EVENT_INT_ST	RMT_CH1_TX_THR_EVENT_INT_ST	RMT_CH0_TX_THR_EVENT_INT_ST	RMT_CH7_ERR_INT_ST	RMT_CH7_RX_END_INT_ST	RMT_CH7_TX_END_INT_ST	RMT_CH6_ERR_INT_ST	RMT_CH6_RX_END_INT_ST	RMT_CH6_TX_END_INT_ST	RMT_CH5_ERR_INT_ST	RMT_CH5_RX_END_INT_ST	RMT_CH5_TX_END_INT_ST	RMT_CH4_ERR_INT_ST	RMT_CH4_RX_END_INT_ST	RMT_CH4_TX_END_INT_ST	RMT_CH3_ERR_INT_ST	RMT_CH3_RX_END_INT_ST	RMT_CH3_TX_END_INT_ST	RMT_CH2_ERR_INT_ST	RMT_CH2_RX_END_INT_ST	RMT_CH2_TX_END_INT_ST	RMT_CH1_ERR_INT_ST	RMT_CH1_RX_END_INT_ST	RMT_CH1_TX_END_INT_ST	RMT_CH0_ERR_INT_ST	RMT_CH0_RX_END_INT_ST	RMT_CH0_TX_END_INT_ST	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_THR_EVENT_INT_ST The masked interrupt status bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (RO)

RMT_CH*n*_ERR_INT_ST The masked interrupt status bit for the RMT_CH*n*_ERR_INT interrupt. (RO)

RMT_CH*n*_RX_END_INT_ST The masked interrupt status bit for the RMT_CH*n*_RX_END_INT interrupt. (RO)

RMT_CH*n*_TX_END_INT_ST The masked interrupt status bit for the RMT_CH*n*_TX_END_INT interrupt. (RO)

Register 7.5: RMT_INT_ENA_REG (0x00a8)

RMT_CH7_TX_THR_EVENT_INT_ENA	RMT_CH6_TX_THR_EVENT_INT_ENA	RMT_CH5_TX_THR_EVENT_INT_ENA	RMT_CH4_TX_THR_EVENT_INT_ENA	RMT_CH3_TX_THR_EVENT_INT_ENA	RMT_CH2_TX_THR_EVENT_INT_ENA	RMT_CH1_TX_THR_EVENT_INT_ENA	RMT_CH0_TX_THR_EVENT_INT_ENA	RMT_CH7_ERR_INT_ENA	RMT_CH7_RX_END_INT_ENA	RMT_CH7_TX_END_INT_ENA	RMT_CH6_ERR_INT_ENA	RMT_CH6_RX_END_INT_ENA	RMT_CH6_TX_END_INT_ENA	RMT_CH5_ERR_INT_ENA	RMT_CH5_RX_END_INT_ENA	RMT_CH5_TX_END_INT_ENA	RMT_CH4_ERR_INT_ENA	RMT_CH4_RX_END_INT_ENA	RMT_CH4_TX_END_INT_ENA	RMT_CH3_ERR_INT_ENA	RMT_CH3_RX_END_INT_ENA	RMT_CH3_TX_END_INT_ENA	RMT_CH2_ERR_INT_ENA	RMT_CH2_RX_END_INT_ENA	RMT_CH2_TX_END_INT_ENA	RMT_CH1_ERR_INT_ENA	RMT_CH1_RX_END_INT_ENA	RMT_CH1_TX_END_INT_ENA	RMT_CH0_ERR_INT_ENA	RMT_CH0_RX_END_INT_ENA	RMT_CH0_TX_END_INT_ENA	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_THR_EVENT_INT_ENA The interrupt enable bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (R/W)

RMT_CH*n*_ERR_INT_ENA The interrupt enable bit for the RMT_CH*n*_ERROR_INT interrupt. (R/W)

RMT_CH*n*_RX_END_INT_ENA The interrupt enable bit for the RMT_CH*n*_RX_END_INT interrupt. (R/W)

RMT_CH*n*_TX_END_INT_ENA The interrupt enable bit for the RMT_CH*n*_TX_END_INT interrupt. (R/W)

Register 7.6: RMT_INT_CLR_REG (0x00ac)

RMT_CH7_TX_THR_EVENT_INT_CLR	RMT_CH6_TX_THR_EVENT_INT_CLR	RMT_CH5_TX_THR_EVENT_INT_CLR	RMT_CH4_TX_THR_EVENT_INT_CLR	RMT_CH3_TX_THR_EVENT_INT_CLR	RMT_CH2_TX_THR_EVENT_INT_CLR	RMT_CH1_TX_THR_EVENT_INT_CLR	RMT_CH0_TX_THR_EVENT_INT_CLR	RMT_CH7_ERR_INT_CLR	RMT_CH7_RX_END_INT_CLR	RMT_CH7_TX_END_INT_CLR	RMT_CH6_ERR_INT_CLR	RMT_CH6_RX_END_INT_CLR	RMT_CH6_TX_END_INT_CLR	RMT_CH5_ERR_INT_CLR	RMT_CH5_RX_END_INT_CLR	RMT_CH5_TX_END_INT_CLR	RMT_CH4_ERR_INT_CLR	RMT_CH4_RX_END_INT_CLR	RMT_CH4_TX_END_INT_CLR	RMT_CH3_ERR_INT_CLR	RMT_CH3_RX_END_INT_CLR	RMT_CH3_TX_END_INT_CLR	RMT_CH2_ERR_INT_CLR	RMT_CH2_RX_END_INT_CLR	RMT_CH2_TX_END_INT_CLR	RMT_CH1_ERR_INT_CLR	RMT_CH1_RX_END_INT_CLR	RMT_CH1_TX_END_INT_CLR	RMT_CH0_ERR_INT_CLR	RMT_CH0_RX_END_INT_CLR	RMT_CH0_TX_END_INT_CLR	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_THR_EVENT_INT_CLR Set this bit to clear the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (WO)

RMT_CH*n*_ERR_INT_CLR Set this bit to clear the RMT_CH*n*_ERRINT interrupt. (WO)

RMT_CH*n*_RX_END_INT_CLR Set this bit to clear the RMT_CH*n*_RX_END_INT interrupt. (WO)

RMT_CH*n*_TX_END_INT_CLR Set this bit to clear the RMT_CH*n*_TX_END_INT interrupt. (WO)

Register 7.7: RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-7) (0x00cc+4**n*)

RMT_CARRIER_HIGH_CH*n*	RMT_CARRIER_LOW_CH*n*	
31 16	15 0	
0x00040	0x00040	Reset

RMT_CARRIER_HIGH_CH*n* This field is used to configure the carrier wave high-level duration (in channel clock periods) for channel *n*. (R/W)

RMT_CARRIER_LOW_CH*n* This field is used to configure the carrier wave low-level duration (in channel clock periods) for channel *n*. (R/W)

Register 7.8: RMT_CH*n*_TX_LIM_REG (*n*: 0-7) (0x00ec+4**n*)

(reserved)	RMT_TX_LIM_CH*n*	
31 9	8 0	
0x000000	0x080	Reset

RMT_TX_LIM_CH*n* When channel *n* sends more entries than specified here, it produces a TX_THR_EVENT interrupt. (R/W)

Register 7.9: RMT_APB_CONF_REG (0x00f0)

(reserved)	RMT_MEM_TX_WRAP_EN	
31 2	1	
0x00000000	0	Reset

RMT_MEM_TX_WRAP_EN bit enables wraparound mode: when the transmitter of a channel has reached the end of its memory block, it will resume sending at the start of its memory region. (R/W)

8. PULSE_CNT

8.1 Introduction

The pulse counter module is designed to count the number of rising and/or falling edges of an input signal. Each pulse counter unit has a 16-bit signed counter register and two channels that can be configured to either increment or decrement the counter. Each channel has a signal input that accepts signal edges to be detected, as well as a control input that can be used to enable or disable the signal input. The inputs have optional filters that can be used to discard unwanted glitches in the signal.

The pulse counter has eight independent units, referred to as PULSE_CNT_U*n*.

8.2 Functional Description

8.2.1 Architecture

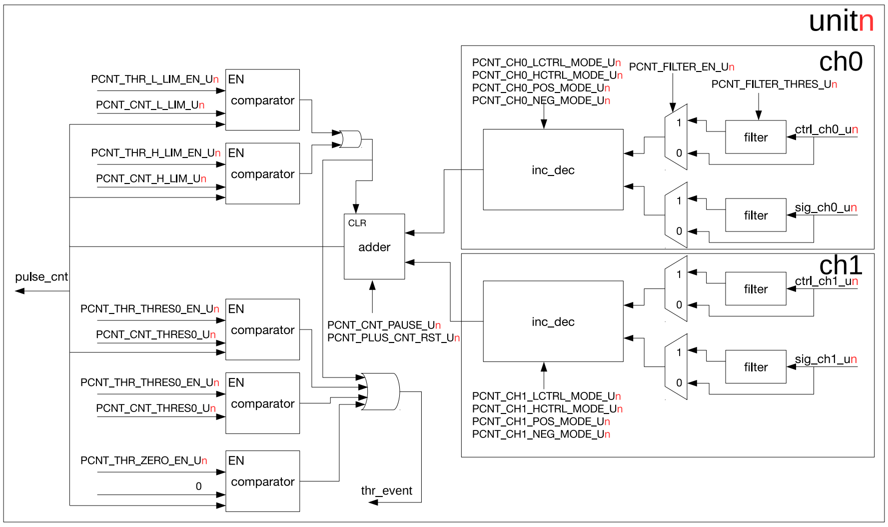


Figure 28: PULSE_CNT Architecture

The architecture of a pulse counter unit is illustrated in Figure 28. Each unit has two channels: ch0 and ch1, which are functionally equivalent. Each channel has a signal input, as well as a control input, which can both be connected to I/O pads. The counting behavior on both the positive and negative edge can be configured separately to increase, decrease, or do nothing to the counter value. Separately, for both control signal levels, the hardware can be configured to modify the edge action: invert it, disable it, or do nothing. The counter itself is a 16-bit signed up/down counter. Its value can be read by software directly, but is also monitored by a set of comparators which can trigger an interrupt.

8.2.2 Counter Channel Inputs

As stated before, the two inputs of a channel can affect the pulse counter in various ways. The specifics of this behaviour are set by LCTRL_MODE and HCTRL_MODE in this case when the control signal is low or high, respectively, and POS_MODE and NEG_MODE for positive and negative edges of the input signal. Setting POS_MODE and NEG_MODE to 1 will increase the counter when an edge is detected, setting them to 2 will decrease the counter and setting at any other value will neutralize the effect of the edge on the counter. LCTR_MODE and HCTR_MODE modify this behaviour, when the control input has the corresponding low or high value: 0 does not modify the NEG_MODE and POS_MODE behaviour, 1 inverts it (setting POS_MODE/NEG_MODE to increase the counter should now decrease the counter and vice versa) and any other value disables counter effects for that signal level.

To summarize, a few examples have been considered. In this table, the effect on the counter for a rising edge is shown for both a low and a high control signal, as well as various other configuration options. For clarity, a short description in brackets is added after the values. Note: x denotes 'do not care'.

POS_ MODE	LCTRL_ MODE	HCTRL_ MODE	sig l→h when ctrl=0	sig l→h when ctrl=1
1 (inc)	0 (-)	0 (-)	Inc ctr	Inc ctr
2 (dec)	0 (-)	0 (-)	Dec ctr	Dec ctr
0 (-)	x	x	No action	No action
1 (inc)	0 (-)	1 (inv)	Inc ctr	Dec ctr
1 (inc)	1 (inv)	0 (-)	Dec ctr	Inc ctr
2 (dec)	0 (-)	1 (inv)	Dec ctr	Inc ctr
1 (inc)	0 (-)	2 (dis)	Inc ctr	No action
1 (inc)	2 (dis)	0 (-)	No action	Inc ctr

This table is also valid for negative edges (sig h→l) on substituting NEG_MODE for POS_MODE.

Each pulse counter unit also features a filter on each of the four inputs, adding the option to ignore short glitches in the signals. If a PCNT_FILTER_EN_U*n* can be set to filter the four input signals of the unit. If this filter is enabled, any pulses shorter than REG_FILTER_THRES_U*n* number of APB_CLK clock cycles will be filtered out and will have no effect on the counter. With the filter disabled, in theory infinitely small glitches could possibly trigger pulse counter action. However, in practice the signal inputs are sampled on APB_CLK edges and even with the filter disabled, pulse widths lasting shorter than one APB_CLK cycle may be missed.

Apart from the input channels, software also has some control over the counter. In particular, the counter value can be frozen to the current value by configuring PCNT_CNT_PAUSE_U*n*. It can also be reset to 0 by configuring PCNT_PULSE_CNT_RST_U*n*.

8.2.3 Watchpoints

The pulse counters have five watchpoints that share one interrupt. Interrupt generation can be enabled or disabled for each individual watchpoint. The watchpoints are:

- Maximum count value: Triggered when PULSE_CNT >= PCNT_THR_H_LIM_U*n*. Additionally, this will reset the counter to 0.
- Minimum count value: Triggered when PULSE_CNT <= PCNT_THR_L_LIM_U*n*. Additionally, this will reset the counter to 0. This is most useful when PCNT_THR_L_LIM_U*n* is set to a negative number.
- Two threshold values: Triggered when PULSE_CNT = PCNT_THR_THRES0_U*n* or PCNT_THR_THRES1_U*n*.
- Zero: Triggered when PULSE_CNT = 0.

8.2.4 Examples

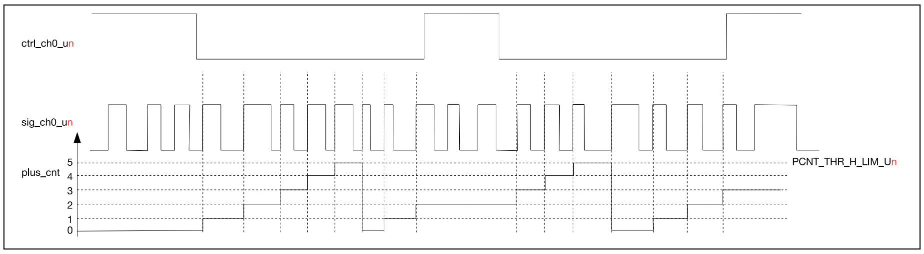


Figure 29: PULSE_CNT Upcounting Diagram

Figure 29 shows channel 0 being used as an up-counter. The configuration of channel 0 is shown below.

- CNT_CH0_POS_MODE_U*n* = 1: increase counter on the rising edge of sig_ch0_u*n*.
- PCNT_CH0_NEG_MODE_U*n* = 0: no counting on the falling edge of sig_ch0_u*n*.
- PCNT_CH0_LCTRL_MODE_U*n* = 0: Do not modify counter mode when sig_ch0_u*n* is low.
- PCNT_CH0_HCTRL_MODE_U*n* = 2: Do not allow counter increments/decrements when sig_ch0_u*n* is high.
- PCNT_THR_H_LIM_U*n* = 5: PULSE_CNT resets to 0 when the count value increases to 5.

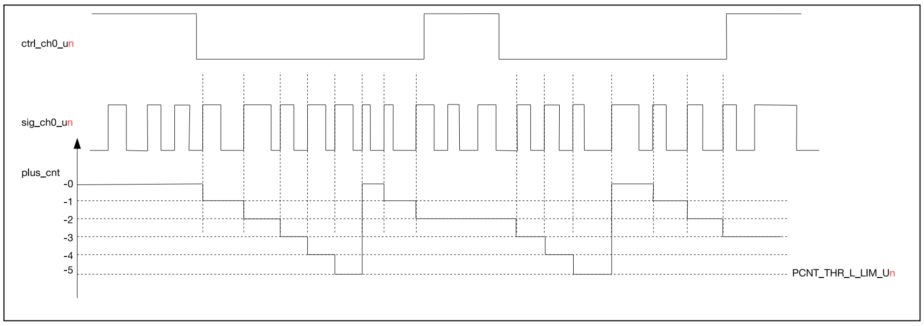


Figure 30: PULSE_CNT Downcounting Diagram

Figure 30 shows channel 0 decrementing the counter. The configuration of channel 0 differs from that in Figure 29 in the following two aspects:

- PCNT_CH0_LCTRL_MODE_U*n* = 1: invert counter mode when ctrl_ch0_u*n* is at low level, so it will decrease, rather than increase, the counter.
- PCNT_THR_H_LIM_U*n* = -5: PULSE_CNT resets to 0 when the count value decreases to -5.

8.2.5 Interrupts

PCNT_CNT_THR_EVENT_U*n*_INT: This interrupt gets triggered when one of the five channel comparators detects a match.

8.3 Register Summary

Name	Description	Address	Access
Configuration registers			

Name	Description	Address	Access
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x3FF57000	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x3FF5700C	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x3FF57018	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x3FF57024	R/W
PCNT_U4_CONF0_REG	Configuration register 0 for unit 4	0x3FF57030	R/W
PCNT_U5_CONF0_REG	Configuration register 0 for unit 5	0x3FF5703C	R/W
PCNT_U6_CONF0_REG	Configuration register 0 for unit 6	0x3FF57048	R/W
PCNT_U7_CONF0_REG	Configuration register 0 for unit 7	0x3FF57054	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x3FF57004	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x3FF57010	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x3FF5701C	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x3FF57028	R/W
PCNT_U4_CONF1_REG	Configuration register 1 for unit 4	0x3FF57034	R/W
PCNT_U5_CONF1_REG	Configuration register 1 for unit 5	0x3FF57040	R/W
PCNT_U6_CONF1_REG	Configuration register 1 for unit 6	0x3FF5704C	R/W
PCNT_U7_CONF1_REG	Configuration register 1 for unit 7	0x3FF57058	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x3FF57008	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x3FF57014	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x3FF57020	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x3FF5702C	R/W
PCNT_U4_CONF2_REG	Configuration register 2 for unit 4	0x3FF57038	R/W
PCNT_U5_CONF2_REG	Configuration register 2 for unit 5	0x3FF57044	R/W
PCNT_U6_CONF2_REG	Configuration register 2 for unit 6	0x3FF57050	R/W
PCNT_U7_CONF2_REG	Configuration register 2 for unit 7	0x3FF5705C	R/W
Counter values			
PCNT_U0_CNT_REG	Counter value for unit 0	0x3FF57060	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x3FF57064	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x3FF57068	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x3FF5706C	RO
PCNT_U4_CNT_REG	Counter value for unit 4	0x3FF57070	RO
PCNT_U5_CNT_REG	Counter value for unit 5	0x3FF57074	RO
PCNT_U6_CNT_REG	Counter value for unit 6	0x3FF57078	RO
PCNT_U7_CNT_REG	Counter value for unit 7	0x3FF5707C	RO
Control registers			
PCNT_CTRL_REG	Control register for all counters	0x3FF570B0	R/W
Interrupt registers			
PCNT_INT_RAW_REG	Raw interrupt status	0x3FF57080	RO
PCNT_INT_ST_REG	Masked interrupt status	0x3FF57084	RO
PCNT_INT_ENA_REG	Interrupt enable bits	0x3FF57088	R/W
PCNT_INT_CLR_REG	Interrupt clear bits	0x3FF5708C	WO

8.4 Registers

Register 8.1: PCNT_U*n*_CONF0_REG (*n*: 0-7) (0x0+0x0Cn*)**

PCNT_CH1_LCTRL_MODE_U*n*	PCNT_CH1_HCTRL_MODE_U*n*	PCNT_CH1_POS_MODE_U*n*	PCNT_CH1_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_NEG_MODE_U*n*	PCNT_THR_THRES1_EN_U*n*	PCNT_THR_THRES0_EN_U*n*	PCNT_THR_L_LIM_EN_U*n*	PCNT_THR_H_LIM_EN_U*n*	PCNT_THR_ZERO_EN_U*n*	PCNT_FILTER_EN_U*n*	PCNT_FILTER_THRES_U*n*	
31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15	14	13	12	11	10	9 0	
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0x010	Reset

PCNT_CH1_LCTRL_MODE_U*n* This register configures how the CH1_POS_MODE/CH1_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH1_HCTRL_MODE_U*n* This register configures how the CH1_POS_MODE/CH1_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH1_POS_MODE_U*n* This register sets the behaviour when the signal input of channel 1 detects a positive edge. (R/W) 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter

PCNT_CH1_NEG_MODE_U*n* This register sets the behaviour when the signal input of channel 1 detects a negative edge. (R/W) 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter

PCNT_CH0_LCTRL_MODE_U*n* This register configures how the CH0_POS_MODE/CH0_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH0_HCTRL_MODE_U*n* This register configures how the CH0_POS_MODE/CH0_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH0_POS_MODE_U*n* This register sets the behaviour when the signal input of channel 0 detects a positive edge. (R/W) 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter

PCNT_CH0_NEG_MODE_U*n* This register sets the behaviour when the signal input of channel 0 detects a negative edge. (R/W) 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter

PCNT_THR_THRES1_EN_U*n* This is the enable bit for unit *n*'s thres1 comparator. (R/W)

PCNT_THR_THRES0_EN_U*n* This is the enable bit for unit *n*'s thres0 comparator. (R/W)

PCNT_THR_L_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_l_lim comparator. (R/W)

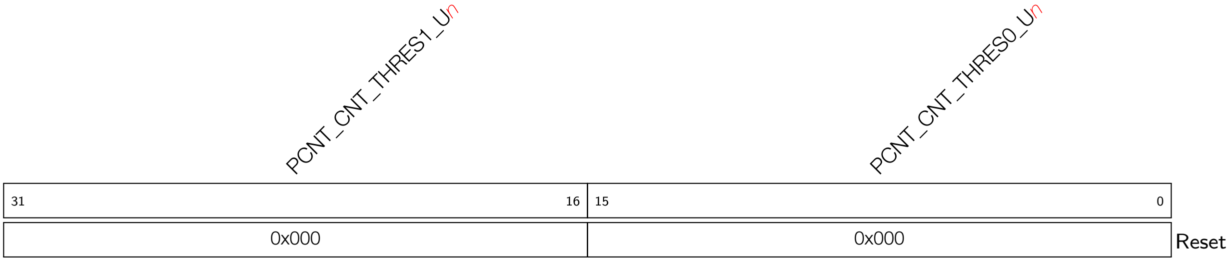
PCNT_THR_H_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_h_lim comparator. (R/W)

PCNT_THR_ZERO_EN_U*n* This is the enable bit for unit *n*'s zero comparator. (R/W)

PCNT_FILTER_EN_U*n* This is the enable bit for unit *n*'s input filter. (R/W)

PCNT_FILTER_THRES_U*n* This sets the maximum threshold, in APB_CLK cycles, for the filter. Any pulses lasting shorter than this will be ignored when the filter is enabled. (R/W)

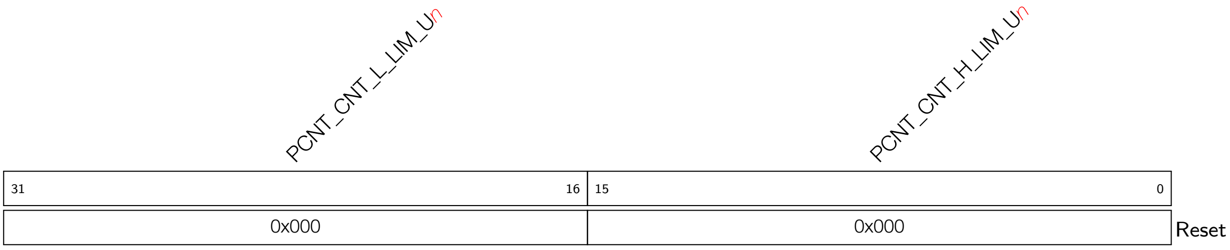
Register 8.2: PCNT_Un_CONF1_REG (n: 0-7) (0x4+0x0C*n)



PCNT_CNT_THRES1_Un This register is used to configure the thres1 value for unit n. (R/W)

PCNT_CNT_THRES0_Un This register is used to configure the thres0 value for unit n. (R/W)

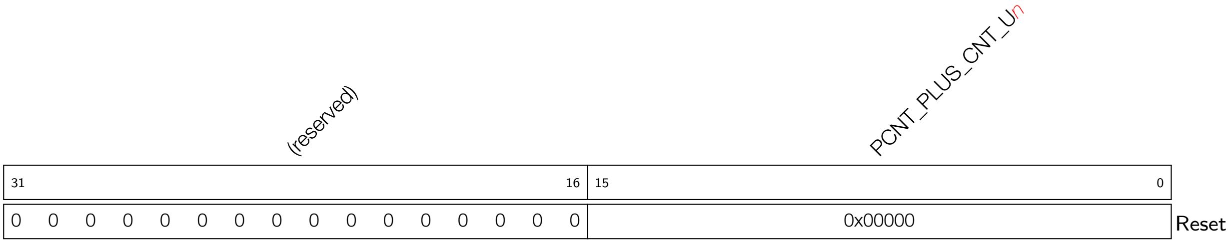
Register 8.3: PCNT_Un_CONF2_REG (n: 0-7) (0x8+0x0C*n)



PCNT_CNT_L_LIM_Un This register is used to configure the thr_l_lim value for unit n. (R/W)

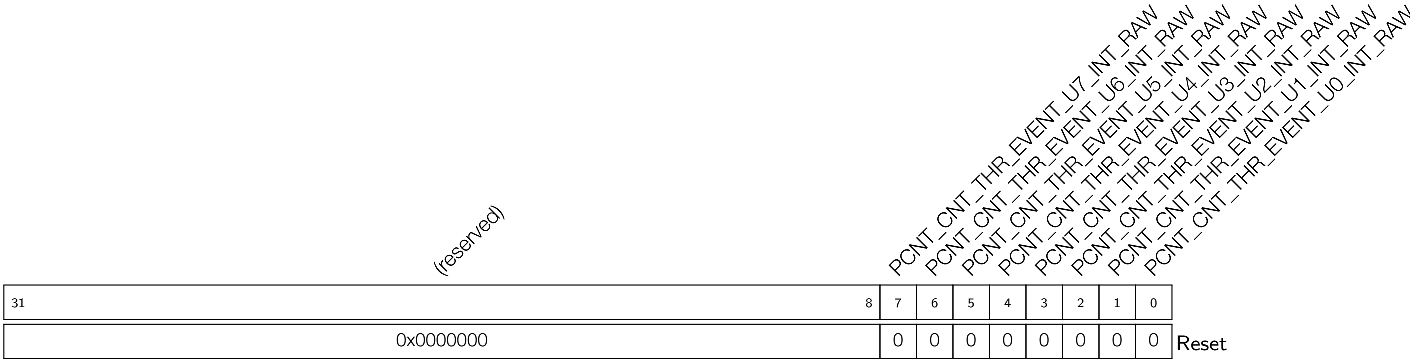
PCNT_CNT_H_LIM_Un This register is used to configure the thr_h_lim value for unit n. (R/W)

Register 8.4: PCNT_Un_CNT_REG (n: 0-7) (0x28+0x0C*n)



PCNT_PLUS_CNT_Un This register stores the current pulse count value for unit n. (RO)

Register 8.5: PCNT_INT_RAW_REG (0x0080)

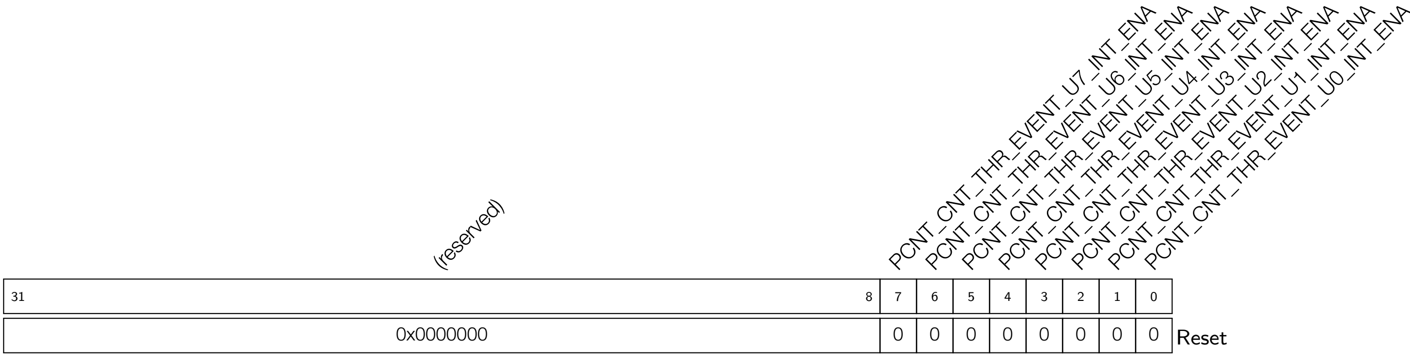


PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 8.6: PCNT_INT_ST_REG (0x0084)

PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 8.7: PCNT_INT_ENA_REG (0x0088)



PCNT_CNT_THR_EVENT_U*n*_INT_ENA The interrupt enable bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 8.8: PCNT_INT_CLR_REG (0x008c)

(reserved)	PCNT_CNT_THR_EVENT_U7_INT_CLR	PCNT_CNT_THR_EVENT_U6_INT_CLR	PCNT_CNT_THR_EVENT_U5_INT_CLR	PCNT_CNT_THR_EVENT_U4_INT_CLR	PCNT_CNT_THR_EVENT_U3_INT_CLR	PCNT_CNT_THR_EVENT_U2_INT_CLR	PCNT_CNT_THR_EVENT_U1_INT_CLR	PCNT_CNT_THR_EVENT_U0_INT_CLR	
31 8	7	6	5	4	3	2	1	0	
0x0000000	0	0	0	0	0	0	0	0	Reset

PCNT_CNT_THR_EVENT_U*n*_INT_CLR Set this bit to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

Register 8.9: PCNT_CTRL_REG (0x00b0)

(reserved)	(reserved)	PCNT_CNT_PAUSE_U7	PCNT_PLUS_CNT_RST_U7	PCNT_CNT_PAUSE_U6	PCNT_PLUS_CNT_RST_U6	PCNT_CNT_PAUSE_U5	PCNT_PLUS_CNT_RST_U5	PCNT_CNT_PAUSE_U4	PCNT_PLUS_CNT_RST_U4	PCNT_CNT_PAUSE_U3	PCNT_PLUS_CNT_RST_U3	PCNT_CNT_PAUSE_U2	PCNT_PLUS_CNT_RST_U2	PCNT_CNT_PAUSE_U1	PCNT_PLUS_CNT_RST_U1	PCNT_CNT_PAUSE_U0	PCNT_PLUS_CNT_RST_U0	
31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x0000	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	Reset

PCNT_CNT_PAUSE_U*n* Set this bit to freeze unit *n*'s counter. (R/W)

PCNT_PLUS_CNT_RST_U*n* Set this bit to clear unit *n*'s counter. (R/W)

9. 64-bit Timers

9.1 Introduction

There are four general-purpose timers embedded in the ESP32. They are all 64-bit generic timers based on 16-bit prescalers and 64-bit auto-reload-capable up/downcounters.

The ESP32 contains two timer modules, each containing two timers. The two timers in a block are indicated by an *x* in TIMG*n*_T*x*; the blocks themselves are indicated by an *n*.

The timers feature:

- A 16-bit clock prescaler, from 2 to 65536
- A 64-bit time-base counter
- Configurable up/down time-base counter: incrementing or decrementing
- Halt and resume of time-base counter
- Auto-reload at alarm
- Software-controlled instant reload
- Level and edge interrupt generation

9.2 Functional Description

9.2.1 16-bit Prescaler

Each timer uses the APB clock (APB_CLK, normally 80 MHz) as the basic clock. This clock is then divided down by a 16-bit precaler which generates the time-base counter clock (TB_clk). Every cycle of TB_clk causes the time-base counter to increment or decrement by one. The timer must be disabled (TIMG*n*_T*x*_EN is cleared) before changing the prescaler divisor which is configured by TIMG*n*_T*x*_DIVIDER register; changing it on an enabled timer can lead to unpredictable results. The prescaler can divide the APB clock by a factor from 2 to 65536. Specifically, when TIMG*n*_T*x*_DIVIDER is either 1 or 2, the clock divisor is 2; when TIMG*n*_T*x*_DIVIDER is 0, the clock divisor is 65536. Any other value will cause the clock to be divided by exactly that value.

9.2.2 64-bit Time-base Counter

The 64-bit time-base counter can be configured to count either up or down, depending on whether TIMG*n*_T*x*_INCREASE is set or cleared, respectively. It supports both auto-reload and software instant reload. An alarm event can be set when the counter reaches a value specified by the software.

Counting can be enabled and disabled by setting and clearing TIMG*n*_T*x*_EN. Clearing this bit essentially freezes the counter, causing it to neither count up nor count down; instead, it retains its value until TIMG*n*_T*x*_EN is set again. Reloading the counter when TIMG*n*_T*x*_EN is cleared will change its value, but counting will not be resumed until TIMG*n*_T*x*_EN is set.

Software can set a new counter value by setting registers TIMG*n*_T*x*_LOAD_LO and TIMG*n*_T*x*_LOAD_HI to the intended new value. The hardware will ignore these register settings until a reload; a reload will cause the contents of these registers to be copied to the counter itself. A reload event can be triggered by an alarm (auto-reload at alarm) or by software (software instant reload). To enable auto-reload at alarm, the register

TIMG*n*_T*x*_AUTORELOAD should be set. If auto-reload at alarm is not enabled, the time-base counter will continue incrementing or decrementing after the alarm. To trigger a software instant reload, any value can be written to the register TIMG*n*_T*x*_LOAD_REG; this will cause the counter value to change instantly. Software can also change the direction of the time-base counter instantly by changing the value of TIMG*n*_T*x*_INCREASE.

The time-base counter can also be read by software, but because the counter is 64-bit, the CPU can only get the value as two 32-bit values, the counter value needs to be latched onto TIMG*n*_T*x*LO_REG and TIMG*n*_T*x*HI_REG first. This is done by writing any value to TIMG*n*_T*x*UPDATE_REG; this will instantly latch the 64-bit timer value onto the two registers. Software can then read them at any point in time. This approach stops the timer value being read erroneously when a carry-over happens between reading the low and high word of the timer value.

9.2.3 Alarm Generation

The timer can trigger an alarm, which can cause a reload and/or an interrupt to occur. The alarm is triggered when the alarm registers TIMG*n*_T*x*_ALARMLO_REG and TIMG*n*_T*x*_ALARMHI_REG match the current timer value. In order to simplify the scenario where these registers are set 'too late' and the counter has already passed these values, the alarm also triggers when the current timer value is higher (for an up-counting timer) or lower (for a down-counting timer) than the current alarm value: if this is the case, the alarm will be triggered immediately upon loading the alarm registers.

9.2.4 MWDT

Each timer module also contains a Main System Watchdog Timer and its associated registers. While these registers are described here, their functional description can be found in the chapter entitled Watchdog Timer.

9.2.5 Interrupts

- TIMG*n*_T*x*_INT_WDT_INT: Generated when a watchdog timer interrupt stage times out.
- TIMG*n*_T*x*_INT_T1_INT: An alarm event on timer 1 generates this interrupt.
- TIMG*n*_T*x*_INT_T0_INT: An alarm event on timer 0 generates this interrupt.

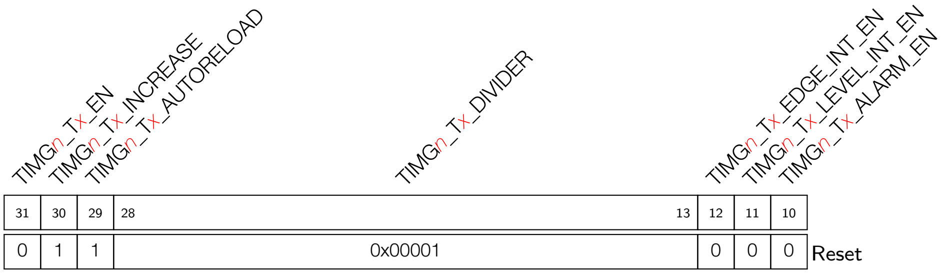
9.3 Register Summary

Name	Description	TIMG0	TIMG1	Acc
Timer 0 configuration and control registers				
TIMG*n*_T0CONFIG_REG	Timer 0 configuration register	0x3FF5F000	0x3FF60000	R/W
TIMG*n*_T0LO_REG	Timer 0 current value, low 32 bits	0x3FF5F004	0x3FF60004	RO
TIMG*n*_T0HI_REG	Timer 0 current value, high 32 bits	0x3FF5F008	0x3FF60008	RO
TIMG*n*_T0UPDATE_REG	Write to copy current timer value to TIMG*n*_T0_(LO/HI)_REG	0x3FF5F00C	0x3FF6000C	WO
TIMG*n*_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x3FF5F010	0x3FF60010	R/W
TIMG*n*_T0ALARMHI_REG	Timer 0 alarm value, high bits	0x3FF5F014	0x3FF60014	R/W
TIMG*n*_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x3FF5F018	0x3FF60018	R/W

Name	Description	TIMG0	TIMG1	Acc
TIMG*n*_T0LOAD_REG	Write to reload timer from TIMG*n*_T0_(LOADLOLOADHI)_REG	0x3FF5F020	0x3FF60020	WO
Timer 1 configuration and control registers				
TIMG*n*_T1CONFIG_REG	Timer 1 configuration register	0x3FF5F024	0x3FF60024	R/W
TIMG*n*_T1LO_REG	Timer 1 current value, low 32 bits	0x3FF5F028	0x3FF60028	RO
TIMG*n*_T1HI_REG	Timer 1 current value, high 32 bits	0x3FF5F02C	0x3FF6002C	RO
TIMG*n*_T1UPDATE_REG	Write to copy current timer value to TIMG*n*_T1_(LO/HI)_REG	0x3FF5F030	0x3FF60030	WO
TIMG*n*_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x3FF5F034	0x3FF60034	R/W
TIMG*n*_T1ALARMHI_REG	Timer 1 alarm value, high 32 bits	0x3FF5F038	0x3FF60038	R/W
TIMG*n*_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x3FF5F03C	0x3FF6003C	R/W
TIMG*n*_T1LOAD_REG	Write to reload timer from TIMG*n*_T1_(LOADLOLOADHI)_REG	0x3FF5F044	0x3FF60044	WO
System watchdog timer configuration and control registers				
TIMG*n*_T*x*_WDTCONFIG0_REG	Watchdog timer configuration register	0x3FF5F048	0x3FF60048	R/W
TIMG*n*_T*x*_WDTCONFIG1_REG	Watchdog timer prescaler register	0x3FF5F04C	0x3FF6004C	R/W
TIMG*n*_T*x*_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x3FF5F050	0x3FF60050	R/W
TIMG*n*_T*x*_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x3FF5F054	0x3FF60054	R/W
TIMG*n*_T*x*_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x3FF5F058	0x3FF60058	R/W
TIMG*n*_T*x*_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x3FF5F05C	0x3FF6005C	R/W
TIMG*n*_T*x*_WDTFEED_REG	Write to feed the watchdog timer	0x3FF5F060	0x3FF60060	WO
TIMG*n*_T*x*_WDTWPROTECT_REG	Watchdog write protect register	0x3FF5F064	0x3FF60064	R/W
Interrupt registers				
TIMG*n*_T*x*_INT_RAW_REG	Raw interrupt status	0x3FF5F09C	0x3FF6009C	RO
TIMG*n*_T*x*_INT_ST_REG	Masked interrupt status	0x3FF5F0A0	0x3FF600A0	RO
TIMG*n*_T*x*_INT_ENA_REG	Interrupt enable bits	0x3FF5F098	0x3FF60098	R/W
TIMG*n*_T*x*_INT_CLR_REG	Interrupt clear bits	0x3FF5F0A4	0x3FF600A4	WO

9.4 Registers

Register 9.1: TIMG*n*_T*x*CONFIG_REG (*x*: 0-1) (0x0+0x24x*)**



TIMG*n*_T*x*_EN When set, the timer *x* time-base counter is enabled. (R/W)

TIMG*n*_T*x*_INCREASE When set, the timer *x* time-base counter will increment every clock tick. When cleared, the timer *x* time-base counter will decrement. (R/W)

TIMG*n*_T*x*_AUTORELOAD When set, timer *x* auto-reload at alarm is enabled. (R/W)

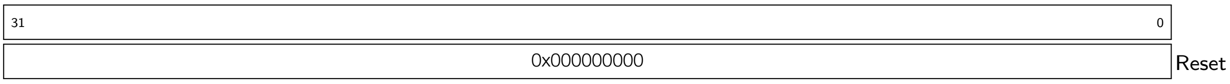
TIMG*n*_T*x*_DIVIDER Timer *x* clock (T*x*_clk) prescale value. (R/W)

TIMG*n*_T*x*_EDGE_INT_EN When set, an alarm will generate an edge type interrupt. (R/W)

TIMG*n*_T*x*_LEVEL_INT_EN When set, an alarm will generate a level type interrupt. (R/W)

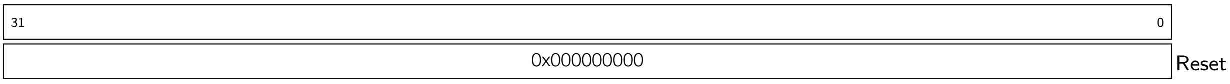
TIMG*n*_T*x*_ALARM_EN When set, the alarm is enabled. (R/W)

Register 9.2: TIMG*n*_T*x*LO_REG (*x*: 0-1) (0x4+0x24x*)**



TIMG*n*_T*x*LO_REG After writing to TIMG*n*_T*x*UPDATE_REG, the low 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 9.3: TIMG*n*_T*x*HI_REG (*x*: 0-1) (0x8+0x24x*)**



TIMG*n*_T*x*HI_REG After writing to TIMG*n*_T*x*UPDATE_REG, the high 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 9.4: TIMG*n*_T*x*UPDATE_REG (*x*: 0-1) (0xC+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*UPDATE_REG Write any value to trigger a timer *x* time-base counter value update (timer *x* current value will be stored in registers above). (WO)

Register 9.5: TIMG*n*_T*x*ALARMLO_REG (*x*: 0-1) (0x10+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*ALARMLO_REG Timer *x* alarm trigger time-base counter value, low 32 bits. (R/W)

Register 9.6: TIMG*n*_T*x*ALARMHI_REG (*x*: 0-1) (0x14+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*ALARMHI_REG Timer *x* alarm trigger time-base counter value, high 32 bits. (R/W)

Register 9.7: TIMG*n*_T*x*LOADLO_REG (*x*: 0-1) (0x18+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*LOADLO_REG Low 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 9.8: TIMG*n*_T*x*LOADHI_REG (*x*: 0-1) (0x1C+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*LOADHI_REG High 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 9.9: TIMG*n*_T*x*LOAD_REG (*x*: 0-1) (0x20+0x24**x*)

31 0	
0x000000000	Reset

TIMG*n*_T*x*LOAD_REG Write any value to trigger a timer *x* time-base counter reload. (WO)

Register 9.10: TIMG*n*_T*x*_WDTCONFIG0_REG (0x0048)

TIMG*n*_T*x*_WDT_EN	TIMG*n*_T*x*_WDT_STG0	TIMG*n*_T*x*_WDT_STG1	TIMG*n*_T*x*_WDT_STG2	TIMG*n*_T*x*_WDT_STG3	TIMG*n*_T*x*_WDT_EDGE_INT_EN	TIMG*n*_T*x*_WDT_LEVEL_INT_EN	TIMG*n*_T*x*_WDT_CPU_RESET_LENGTH	TIMG*n*_T*x*_WDT_SYS_RESET_LENGTH	TIMG*n*_T*x*_WDT_FLASHBOOT_MOD_EN	
31	30 29	28 27	26 25	24 23	22	21	20 18	17 15	14	
0	0	0	0	0	0	0	0x1	0x1	1	Reset

TIMG*n*_T*x*_WDT_EN When set, MWDT is enabled. (R/W)

TIMG*n*_T*x*_WDT_STG0 Stage 0 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_STG1 Stage 1 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_STG2 Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_STG3 Stage 3 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_EDGE_INT_EN When set, an edge type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

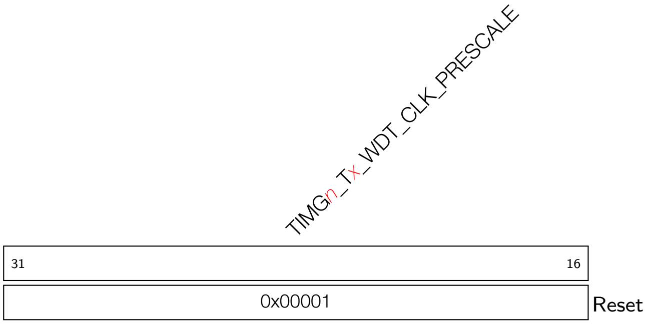
TIMG*n*_T*x*_WDT_LEVEL_INT_EN When set, a level type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

TIMG*n*_T*x*_WDT_CPU_RESET_LENGTH CPU reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 μs, 7: 3.2 μs. (R/W)

TIMG*n*_T*x*_WDT_SYS_RESET_LENGTH System reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 μs, 7: 3.2 μs. (R/W)

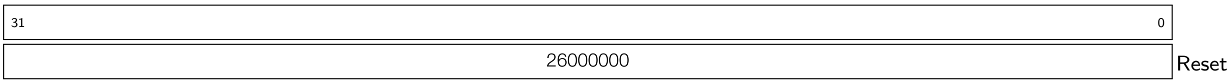
TIMG*n*_T*x*_WDT_FLASHBOOT_MOD_EN When set, Flash boot protection is enabled. (R/W)

Register 9.11: TIMG*n*_T*x*_WDTCONFIG1_REG (0x004c)



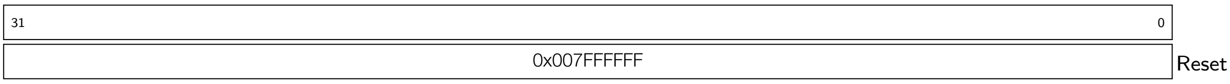
TIMG*n*_T*x*_WDT_CLK_PRESCALE MWDT clock prescale value. MWDT clock period = 12.5 ns * TIMG*n*_T*x*_WDT_CLK_PRESCALE. (R/W)

Register 9.12: TIMG*n*_T*x*_WDTCONFIG2_REG (0x0050)



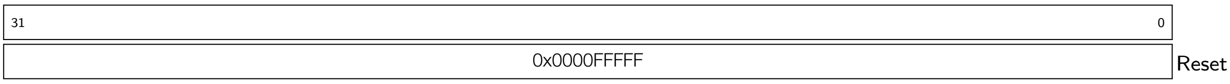
TIMG*n*_T*x*_WDTCONFIG2_REG Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 9.13: TIMG*n*_T*x*_WDTCONFIG3_REG (0x0054)



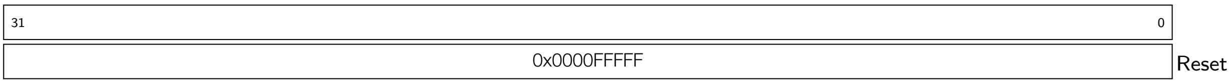
TIMG*n*_T*x*_WDTCONFIG3_REG Stage 1 timeout value, in MWDT clock cycles. (R/W)

Register 9.14: TIMG*n*_T*x*_WDTCONFIG4_REG (0x0058)



TIMG*n*_T*x*_WDTCONFIG4_REG Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 9.15: TIMG*n*_T*x*_WDTCONFIG5_REG (0x005c)



TIMG*n*_T*x*_WDTCONFIG5_REG Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 9.16: TIMG*n*_T*x*_WDTFEED_REG (0x0060)

31 ... 0	
0x000000000	Reset

TIMG*n*_T*x*_WDTFEED_REG Write any value to feed the MWDT. (WO)

Register 9.17: TIMG*n*_T*x*_WDTWPROTECT_REG (0x0064)

31 ... 0	
0x050D83AA1	Reset

TIMG*n*_T*x*_WDTWPROTECT_REG If the register contains a different value than its reset value, write protection is enabled. (R/W)

Register 9.18: TIMG*n*_T*x*_INT_ENA_REG (0x0098)

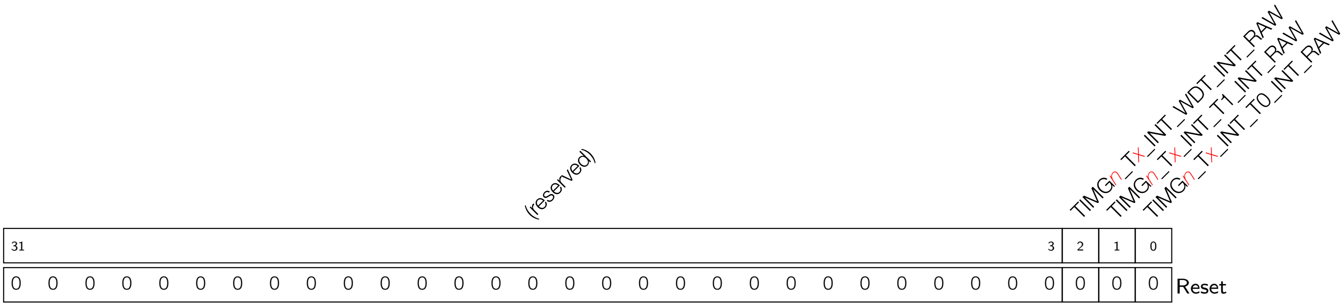
(reserved)	TIMG*n*_T*x*_INT_WDT_INT_ENA	TIMG*n*_T*x*_INT_T1_INT_ENA	TIMG*n*_T*x*_INT_T0_INT_ENA	
31 ... 3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	Reset

TIMG*n*_T*x*_INT_WDT_INT_ENA The interrupt enable bit for the TIMG*n*_T*x*_INT_WDT_INT interrupt. (R/W) (R/W)

TIMG*n*_T*x*_INT_T1_INT_ENA The interrupt enable bit for the TIMG*n*_T*x*_INT_T1_INT interrupt. (R/W) (R/W)

TIMG*n*_T*x*_INT_T0_INT_ENA The interrupt enable bit for the TIMG*n*_T*x*_INT_T0_INT interrupt. (R/W) (R/W)

Register 9.19: TIMG*n*_T*x*_INT_RAW_REG (0x009c)

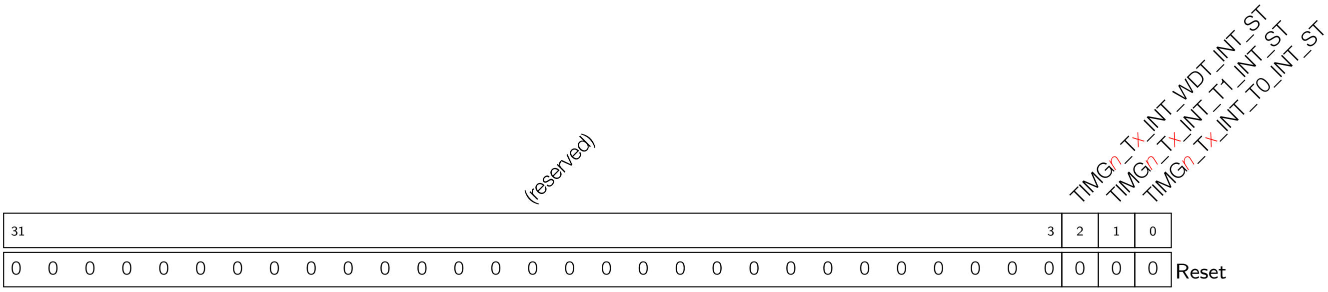


TIMG*n*_T*x*_INT_WDT_INT_RAW The raw interrupt status bit for the TIMG*n*_T*x*_INT_WDT_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T1_INT_RAW The raw interrupt status bit for the TIMG*n*_T*x*_INT_T1_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T0_INT_RAW The raw interrupt status bit for the TIMG*n*_T*x*_INT_T0_INT interrupt. (RO)

Register 9.20: TIMG*n*_T*x*_INT_ST_REG (0x00a0)

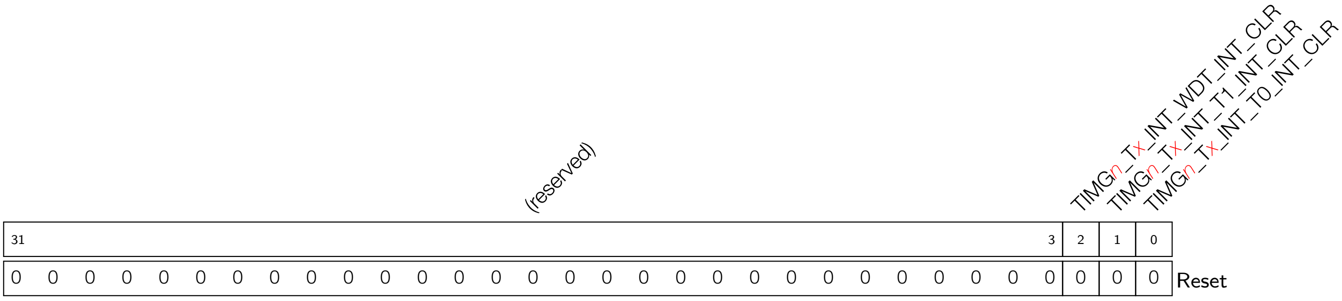


TIMG*n*_T*x*_INT_WDT_INT_ST The masked interrupt status bit for the TIMG*n*_T*x*_INT_WDT_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T1_INT_ST The masked interrupt status bit for the TIMG*n*_T*x*_INT_T1_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T0_INT_ST The masked interrupt status bit for the TIMG*n*_T*x*_INT_T0_INT interrupt. (RO)

Register 9.21: TIMG*n*_T*x*_INT_CLR_REG (0x00a4)



TIMG*n*_T*x*_INT_WDT_INT_CLR Set this bit to clear the TIMG*n*_T*x*_INT_WDT_INT interrupt. (WO)

TIMG*n*_T*x*_INT_T1_INT_CLR Set this bit to clear the TIMG*n*_T*x*_INT_T1_INT interrupt. (WO)

TIMG*n*_T*x*_INT_T0_INT_CLR Set this bit to clear the TIMG*n*_T*x*_INT_T0_INT interrupt. (WO)

10. Watchdog Timers

10.1 Introduction

The ESP32 has three watchdog timers: one in each of the two timer modules (called Main System Watchdog Timer, or MWDT) and one in the RTC module (which is called the RTC Watchdog Timer, or RWDT). These watchdog timers are intended to recover from an unforeseen fault, causing the application program to abandon its normal sequence. A watchdog timer has four stages. Each stage may take one out of three or four actions upon the expiry of a programmed period of time for this stage, unless the watchdog is fed or disabled. The actions are: interrupt, CPU reset, core reset and system reset. Only the RWDT can trigger the system reset, and is able to reset the entire chip and the main system including the RTC itself. A timeout value can be set for each stage individually.

During flash boot, the RWDT and the first MWDT start automatically in order to detect and recover from booting problems.

10.2 Features

- Four stages, each of which can be configured or disabled separately
- Programmable time period for each stage
- One out of three or four possible actions (interrupt, CPU reset, core reset and system reset) upon the expiry of each stage
- 32-bit expiry counter
- Write protection, to prevent the RWDT and MWDT configuration from being inadvertently altered.
- Flash boot protection
  If the boot process from an SPI flash does not complete within a predetermined period of time, the watchdog will reboot the entire main system.

10.3 Functional Description

10.3.1 Clock

The RWDT is clocked from the RTC slow clock, which usually will be 32 KHz. The MWDT clock source is derived from the APB clock via a pre-MWDT 16-bit configurable prescaler. For either watchdog, the clock source is fed into the 32-bit expiry counter. When this counter reaches the timeout value of the current stage, the action configured for the stage will execute, the expiry counter will be reset and the next stage will become active.

10.3.1.1 Operating Procedure

When a watchdog timer is enabled, it will proceed in loops from stage 0 to stage 3, then back to stage 0 and start again. The expiry action and time period for each stage can be configured individually.

Every stage can be configured for one of the following actions when the expiry timer reaches the stage's timeout value:

- Trigger an interrupt
  When the stage expires an interrupt is triggered.
- Reset a CPU core
  When the stage expires the designated CPU core will be reset. MWDT0 CPU reset only resets the PRO CPU. MWDT1 CPU reset only resets the APP CPU. The RWDT CPU reset can reset either of them, or both, or none, depending on configuration.
- Reset the main system
  When the stage expires, the main system, including the MWDTs, will be reset. In this article, the main system includes the CPU and all peripherals. The RTC is an exception to this, and it will not be reset.
- Reset the main system and RTC
  When the stage expires the main system and the RTC will both be reset. This action is only available in the RWDT.
- Disabled
  This stage will have no effects on the system.

When software feeds the watchdog timer, it returns to stage 0 and its expiry counter restarts from 0.

10.3.1.2 Write Protection

Both the MWDTs, as well as the RWDT, can be protected from accidental writing. To accomplish this, they have a write-key register (TIMERS_WDT_WKEY for the MWDT, RTC_CNTL_WDT_WKEY for the RWDT.) On reset, these registers are initialized to the value 0x50D83AA1. When the value in this register is changed from 0x50D83AA1, write protection is enabled. Writes to any WDT register, including the feeding register (but excluding the write-key register itself), are ignored. The recommended procedure for accessing a WDT is:

1. Disable the write protection
2. Make the required modification or feed the watchdog
3. Re-enable the write protection

10.3.1.3 Flash Boot Protection

During flash booting, the MWDT in timer group 0 (TIMG0), as well as the RWDT, are automatically enabled. Stage 0 for the enabled MWDT is automatically configured to reset the system upon expiry; stage 0 for the RWDT resets the RTC when it expires. After booting, the register TIMERS_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for the MWDT, and RTC_CNTL_WDT_FLASHBOOT_MOD_EN should be cleared to do the same for the RWDT. After this, the MWDT and RWDT can be configured by software.

10.3.1.4 Registers

The MWDT registers are part of the timer submodule and are described in the Timer Registers section. The RWDT registers are part of the RTC submodule and are described in the RTC Registers section.

11. eFuse Controller

11.1 Introduction

The ESP32 has a number of eFuses which store system parameters. Fundamentally, an eFuse is a single bit of non-volatile memory with the restriction that once an eFuse bit is programmed to 1, it can never be reverted to 0. Software can instruct the eFuse Controller to program each bit for each system parameter as needed.

Some of these system parameters can be read by software using the eFuse Controller. Some of the system parameters are also directly used by hardware modules.

11.2 Features

- Configuration of 26 system parameters
- Optional write-protection
- Optional software-read-protection

11.3 Functional Description

11.3.1 Structure

Twenty-six system parameters with different bit width are stored in the eFuses. The name of each system parameter and the corresponding bit width are shown in Table 28. Among those parameters, efuse_wr_disable, efuse_rd_disable, and coding_scheme are directly used by the eFuse Controller.

Table 28: System Parameter

Name	Bit width	Program -Protection by efuse_wr_disable	Software-Read -Protection by efuse_rd_disable	Description
efuse_wr_disable	16	1	-	controls the eFuse Controller
efuse_rd_disable	4	0	-	controls the eFuse Controller
flash_crypt_cnt	8	2	-	governs the flash encryption/ decryption
WIFI_MAC_Address	56	3	-	Wi-Fi MAC address and CRC
SPI_pad_config_hd	5	3	-	configures the SPI I/O to a certain pad
chip_version	4	3	-	chip version
XPD_SDIO_REG	1	5	-	powers up the flash regulator
SDIO_TIEH	1	5	-	configures the flash regulator voltage: set to 1 for 3.3 V and set to 0 for 1.8 V
sdio_force	1	5	-	determines whether XPD_SDIO_REG and SDIO_TIEH can control the flash regulator

Name	Bit width	Program -Protection by efuse_wr_disable	Software-Read -Protection by efuse_rd_disable	Description
SPI_pad_config_clk	5	6	-	configures the SPI I/O to a certain pad
SPI_pad_config_q	5	6	-	configures the SPI I/O to a certain pad
SPI_pad_config_d	5	6	-	configures the SPI I/O to a certain pad
SPI_pad_config_cs0	5	6	-	configures the SPI I/O to a certain pad
flash_crypt_config	4	10	3	governs flash encryption/ decryption
coding_scheme	2	10	3	controls the eFuse Controller
console_debug_disable	1	15	-	disables serial output from the BootROM when set to 1
abstract_done_0	1	12	-	determines the status of Secure Boot
abstract_done_1	1	13	-	determines the status of Secure Boot
JTAG_disable	1	14	-	disables access to the JTAG controllers so as to effectively disable external use of JTAG
download_dis_encrypt	1	15	-	governs flash encryption/ decryption
download_dis_decrypt	1	15	-	governs flash encryption/ decryption
download_dis_cache	1	15	-	disables cache when boot mode is the Download Mode
key_status	1	10	3	determines whether BLOCK3 is deployed for user purposes
BLOCK1	256/192/128	7	0	governs flash encryption/ decryption
BLOCK2	256/192/128	8	1	key for Secure Boot
BLOCK3	256/192/128	9	2	key for user purposes

11.3.1.1 System Parameter efuse_wr_disable

The system parameter efuse_wr_disable determines whether all of the system parameters are write-protected. Since efuse_wr_disable is a system parameter as well, it also determines whether it itself is write-protected.

If a system parameter is not write-protected, its unprogrammed bits can be programmed from 0 to 1. The bits previously programmed to 1 will remain 1. When a system parameter is write-protected, none of its bits can be programmed: The unprogrammed bits will always remain 0 and the programmed bits will always remain 1.

The write-protection status of each system parameter corresponds to a bit in efuse_wr_disable. When the corresponding bit is set to 0, the system parameter is not write-protected. When the corresponding bit is set to 1, the system parameter is write-protected. If a system parameter is already write-protected, it will remain write-protected. The column entitled ”Program-Protection by efuse_wr_disable” in Table 28 lists the corresponding bits that determine the write-protection status of each system parameter.

11.3.1.2 System Parameter efuse_rd_disable

Of the 26 system parameters, 20 are not constrained by software-read-protection. These are marked by ”-” in the column entitled ”Software-Read-Protection by efuse_rd_disable” in Table 28. Those system parameters, some of which are used by software and hardware modules at the same time, can be read by software via the eFuse Controller at any time.

When not software-read-protected, the other six system parameters can both be read by software and used by hardware modules. When they are software-read-protected, they can only be used by the hardware modules.

The column ”Software-Read-Protection by efuse_rd_disable” in Table 28 lists the corresponding bits in efuse_rd_disable that determine the software read-protection status of the six system parameters. If a bit in the system parameter efuse_rd_disable is 0, the system parameter controlled by the bit is not software-read-protected. If a bit in the system parameter efuse_rd_disable is 1, the system parameter controlled by the bit is software-read-protected. If a system parameter is software-read-protected, it will remain in this state.

11.3.1.3 System Parameter coding_scheme

As Table 29 shows, only three system parameters, BLOCK1, BLOCK2, and BLOCK3, have variable bit width. Their bit width is controlled by another system parameter, coding_scheme. Despite their variable bit width, BLOCK1, BLOCK2, and BLOCK3 are assigned a fixed number of bits in eFuse. There is an encoding mapping between these three system parameters and their corresponding stored values in eFuse. For details please see Table 29.

Table 29: BLOCK1/2/3 Encoding

coding_scheme[1:0]	Width of BLOCK1/2/3	Coding scheme	Number of bits in eFuse
00/11	256	None	256
01	192	3/4	256
10	128	Repeat	256

The three coding schemes are explained as follows:

- $BLOCKN$ represents any of the following three system parameters: BLOCK1, BLOCK2 or BLOCK3.
- $BLOCKN[255:0]$, $BLOCKN[191:0]$, and $BLOCKN[127:0]$ represent each bit of the three system parameters in the three encoding schemes.
- ${}^{e}BLOCKN[255:0]$ represents each corresponding bit of those system parameters in eFuse after being encoded.

None

$$^{e}BLOCKN[255:0] = BLOCKN[255:0]$$

3/4

$$\begin{array}{lll} BLOCKN_i^j[7:0] = BLOCKN[48i+8j+7:48i+8j] & i \in \{0,1,2,3\} & j \in \{0,1,2,3,4,5\} \\ ^{e}BLOCKN_i^j[7:0] = {}^{e}BLOCKN[64i+8j+7:64i+8j] & i \in \{0,1,2,3\} & j \in \{0,1,2,3,4,5,6,7\} \end{array}$$

$$^{e}BLOCKN_i^j[7:0] = \begin{cases} BLOCKN_i^j[7:0] & j \in \{0,1,2,3,4,5\} \\ BLOCKN_i^0[7:0] \oplus BLOCKN_i^1[7:0] \\ \quad \oplus BLOCKN_i^2[7:0] \oplus BLOCKN_i^3[7:0] & j \in \{6\} \\ \quad \oplus BLOCKN_i^4[7:0] \oplus BLOCKN_i^5[7:0] \\ \sum_{l=0}^{5}(l+1)\sum_{k=0}^{7} BLOCKN_i^l[k] & j \in \{7\} \end{cases} \qquad i \in \{0,1,2,3\}$$

$\oplus$ means bitwise XOR

$\sum$ and $+$ mean summation

Repeat

$$^{e}BLOCKN[255:128] = {}^{e}BLOCKN[127:0] = BLOCKN[127:0]$$

11.3.2 Programming of System Parameters

The programming of variable-length system parameters BLOCK1, BLOCK2, and BLOCK3 is different from that of the fixed-length system parameters. **We program the $^{e}BLOCKN[255:0]$ value of encoded system parameters BLOCK1, BLOCK2, and BLOCK3 instead of directly programming the system parameters. The bit width of $^{e}BLOCKN[255:0]$ is always 256.** Fixed-length system parameters, in contrast, are programmed without encoding them first.

Each bit of the 23 fixed-length system parameters and the three encoded variable-length system parameters corresponds to a program register bit, as shown in Table 30. The register bits will be used when programming system parameters.

Table 30: Program Register

System parameter			Register	
Name	Width	Bit	Name	Bit
efuse_wr_disable	16	[15:0]	EFUSE_BLK0_WDATA0_REG	[15:0]
efuse_rd_disable	4	[3:0]		[19:16]
flash_crypt_cnt	8	[7:0]		[27:20]
WIFI_MAC_Address	56	[31:0]	EFUSE_BLK0_WDATA1_REG	[31:0]
		[55:32]	EFUSE_BLK0_WDATA2_REG	[23:0]
SPI_pad_config_hd	5	[4:0]	EFUSE_BLK0_WDATA3_REG	[8:4]
chip_version	4	[3:0]		[12:9]
XPD_SDIO_REG	1	[0]	EFUSE_BLK0_WDATA4_REG	[14]
SDIO_TIEH	1	[0]		[15]
sdio_force	1	[0]		[16]

System parameter			Register	
Name	Width	Bit	Name	Bit
SPI_pad_config_clk	5	[4:0]	EFUSE_BLK0_WDATA5_REG	[4:0]
SPI_pad_config_q	5	[4:0]		[9:5]
SPI_pad_config_d	5	[4:0]		[14:10]
SPI_pad_config_cs0	5	[4:0]		[19:15]
flash_crypt_config	4	[3:0]		[31:28]
coding_scheme	2	[1:0]	EFUSE_BLK0_WDATA6_REG	[1:0]
console_debug_disable	1	[0]		[2]
abstract_done_0	1	[0]		[4]
abstract_done_1	1	[0]		[5]
JTAG_disable	1	[0]		[6]
download_dis_encrypt	1	[0]		[7]
download_dis_decrypt	1	[0]		[8]
download_dis_cache	1	[0]		[9]
key_status	1	[0]		[10]
BLOCK1	256/192/128	[31:0]	EFUSE_BLK1_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK1_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK1_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK1_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK1_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK1_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK1_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK1_WDATA7_REG	[31:0]
BLOCK2	256/192/128	[31:0]	EFUSE_BLK2_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK2_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK2_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK2_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK2_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK2_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK2_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK2_WDATA7_REG	[31:0]
BLOCK3	256/192/128	[31:0]	EFUSE_BLK3_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK3_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK3_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK3_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK3_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK3_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK3_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK3_WDATA7_REG	[31:0]

The process of programming system parameters is as follows:

1. Configure EFUSE_CLK_SEL0 bit, EFUSE_CLK_SEL1 bit of register EFUSE_CLK, and EFUSE_DAC_CLK_DIV bit of register EFUSE_DAC_CONF.
2. Set the corresponding register bit of the system parameter bit to be programmed to 1.

3. Write 0x5A5A into register EFUSE_CONF.
4. Write 0x2 into register EFUSE_CMD.
5. Poll register EFUSE_CMD until it is 0x0, or wait for a program-done interrupt.
6. Write 0x5AA5 into register EFUSE_CONF.
7. Write 0x1 into register EFUSE_CMD.
8. Poll register EFUSE_CMD until it is 0x0, or wait for a read-done interrupt.
9. Set the corresponding register bit of the programmed bit to 0.

The configuration values of the EFUSE_CLK_SEL0 bit, EFUSE_CLK_SEL1 bit of register EFUSE_CLK, and the EFUSE_DAC_CLK_DIV bit of register EFUSE_DAC_CONF are based on the current APB_CLK frequency, as is shown in Table 31.

Table 31: Timing Configuration

Register	Configuration Value	APB_CLK Frequency		
		26 MHz	40 MHz	80 MHz
EFUSE_CLK	EFUSE_CLK_SEL0[7:0]	8'd250	8'd160	8'd80
	EFUSE_CLK_SEL1[7:0]	8'd255	8'd255	8'd128
EFUSE_DAC_CONF	EFUSE_DAC_CLK_DIV[7:0]	8'd52	8'd80	8'd160

The two methods to identify the generation of program/read-done interrupts are as follows:

Method One:

1. Poll bit 1/0 in register EFUSE_INT_RAW until bit 1/0 is 1, which represents the generation of an program/read-done interrupt.
2. Set the bit 1/0 in register EFUSE_INT_CLR to 1 to clear the program/read-done interrupts.

Method Two:

1. Set bit 1/0 in register EFUSE_INT_ENA to 1 to enable eFuse Controller to post a program/read-done interrupt.
2. Configure Interrupt Matrix to enable the CPU to respond to an EFUSE_INT interrupt.
3. A program/read-done interrupt is generated.
4. Read bit 1/0 in register EFUSE_INT_ST to identify the generation of the program/read-done interrupt.
5. Set bit 1/0 in register EFUSE_INT_CLR to 1 to clear the program/read-done interrupt.

The programming of different system parameters and even the programming of different bits of the same system parameter can be completed separately in multiple programmings. It is, however, recommended that users minimize programming cycles, and program all the bits that need to be programmed in a system parameter in one programming action. In addition, after all system parameters controlled by a certain bit of efuse_wr_disable are programmed, that bit should be immediately programmed. The programming of system parameters controlled by a certain bit of efuse_wr_disable, and the programming of that bit can even be completed at the same time. **Repeated programming of programmed bits is strictly forbidden.**

11.3.3 Software Reading of System Parameters

Each bit of the 23 fixed-length system parameters and the three variable-length system parameters corresponds to a software-read register bit, as shown in Table 32. Software can use the value of each system parameter by reading the value in the corresponding register.

The bit width of system parameters BLOCK1, BLOCK2, and BLOCK3 is variable. Although 256 register bits have been assigned to each of the three parameters, as shown in Table 32, some of the 256 register bits are useless in the 3/4 coding and the Repeat coding scheme. In the None coding scheme, the corresponding register bit of each bit of $BLOCKN[255:0]$ is used. In the 3/4 coding scheme, only the corresponding register bits of $BLOCKN[191:0]$ are useful. In Repeat coding scheme, only the corresponding bits of $BLOCKN[127:0]$ are useful. In different coding schemes, the values of useless register bits read by software are invalid. **The values of useful register bits read by software are the system parameters BLOCK1, BLOCK2, and BLOCK3 themselves instead of their values after being encoded.**

Table 32: Software Read Register

System parameter			Register	
Name	Bit Width	Bit	Name	Bit
efuse_wr_disable	16	[15:0]	EFUSE_BLK0_RDATA0_REG	[15:0]
efuse_rd_disable	4	[3:0]		[19:16]
flash_crypt_cnt	8	[7:0]		[27:20]
WIFI_MAC_Address	56	[31:0]	EFUSE_BLK0_RDATA1_REG	[31:0]
		[55:32]	EFUSE_BLK0_RDATA2_REG	[23:0]
SPI_pad_config_hd	5	[4:0]	EFUSE_BLK0_RDATA3_REG	[8:4]
chip_version	4	[3:0]		[12:9]
XPD_SDIO_REG	1	[0]	EFUSE_BLK0_RDATA4_REG	[14]
SDIO_TIEH	1	[0]		[15]
sdio_force	1	[0]		[16]
SPI_pad_config_clk	5	[4:0]	EFUSE_BLK0_RDATA5_REG	[4:0]
SPI_pad_config_q	5	[4:0]		[9:5]
SPI_pad_config_d	5	[4:0]		[14:10]
SPI_pad_config_cs0	5	[4:0]		[19:15]
flash_crypt_config	4	[3:0]		[31:28]
coding_scheme	2	[1:0]	EFUSE_BLK0_RDATA6_REG	[1:0]
console_debug_disable	1	[0]		[2]
abstract_done_0	1	[0]		[4]
abstract_done_1	1	[0]		[5]
JTAG_disable	1	[0]		[6]
download_dis_encrypt	1	[0]		[7]
download_dis_decrypt	1	[0]		[8]
download_dis_cache	1	[0]		[9]
key_status	1	[0]		[10]

System parameter			Register	
Name	Bit Width	Bit	Name	Bit
BLOCK1	256/192/128	[31:0]	EFUSE_BLK1_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK1_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK1_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK1_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK1_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK1_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK1_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK1_RDATA7_REG	[31:0]
BLOCK2	256/192/128	[31:0]	EFUSE_BLK2_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK2_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK2_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK2_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK2_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK2_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK2_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK2_RDATA7_REG	[31:0]
BLOCK3	256/192/128	[31:0]	EFUSE_BLK3_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK3_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK3_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK3_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK3_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK3_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK3_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK3_RDATA7_REG	[31:0]

11.3.4 The Use of System Parameters by Hardware Modules

Hardware modules are directly hardwired to the ESP32 in order to use the system parameters. Software cannot change this behaviour. **Hardware modules use the decoded values of system parameters BLOCK1, BLOCK2, and BLOCK3, not their encoded values.**

11.3.5 Interrupts

- EFUSE_PGM_DONE_INT: Triggered when eFuse programming has finished.
- EFUSE_READ_DONE_INT: Triggered when eFuse reading has finished.

11.4 Register Summary

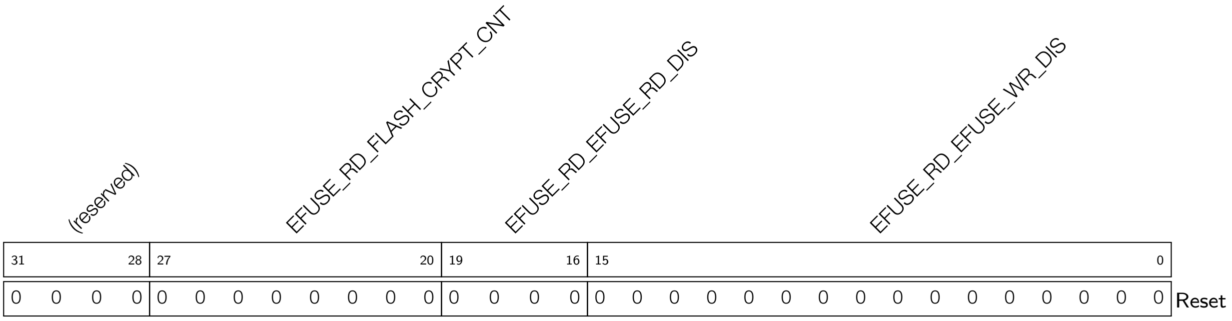
Name	Description	Address	Access
eFuse data read registers			
EFUSE_BLK0_RDATA0_REG	Returns data word 0 in eFuse BLOCK 0	0x3FF5A000	RO
EFUSE_BLK0_RDATA1_REG	Returns data word 1 in eFuse BLOCK 0	0x3FF5A004	RO
EFUSE_BLK0_RDATA2_REG	Returns data word 2 in eFuse BLOCK 0	0x3FF5A008	RO

Name	Description	Address	Access
EFUSE_BLK0_RDATA3_REG	Returns data word 3 in eFuse BLOCK 0	0x3FF5A00C	RO
EFUSE_BLK0_RDATA4_REG	Returns data word 4 in eFuse BLOCK 0	0x3FF5A010	RO
EFUSE_BLK0_RDATA5_REG	Returns data word 5 in eFuse BLOCK 0	0x3FF5A014	RO
EFUSE_BLK0_RDATA6_REG	Returns data word 6 in eFuse BLOCK 0	0x3FF5A018	RO
EFUSE_BLK1_RDATA0_REG	Returns data word 0 in eFuse BLOCK 1	0x3FF5A038	RO
EFUSE_BLK1_RDATA1_REG	Returns data word 1 in eFuse BLOCK 1	0x3FF5A03C	RO
EFUSE_BLK1_RDATA2_REG	Returns data word 2 in eFuse BLOCK 1	0x3FF5A040	RO
EFUSE_BLK1_RDATA3_REG	Returns data word 3 in eFuse BLOCK 1	0x3FF5A044	RO
EFUSE_BLK1_RDATA4_REG	Returns data word 4 in eFuse BLOCK 1	0x3FF5A048	RO
EFUSE_BLK1_RDATA5_REG	Returns data word 5 in eFuse BLOCK 1	0x3FF5A04C	RO
EFUSE_BLK1_RDATA6_REG	Returns data word 6 in eFuse BLOCK 1	0x3FF5A050	RO
EFUSE_BLK1_RDATA7_REG	Returns data word 7 in eFuse BLOCK 1	0x3FF5A054	RO
EFUSE_BLK2_RDATA0_REG	Returns data word 0 in eFuse BLOCK 2	0x3FF5A058	RO
EFUSE_BLK2_RDATA1_REG	Returns data word 1 in eFuse BLOCK 2	0x3FF5A05C	RO
EFUSE_BLK2_RDATA2_REG	Returns data word 2 in eFuse BLOCK 2	0x3FF5A060	RO
EFUSE_BLK2_RDATA3_REG	Returns data word 3 in eFuse BLOCK 2	0x3FF5A064	RO
EFUSE_BLK2_RDATA4_REG	Returns data word 4 in eFuse BLOCK 2	0x3FF5A068	RO
EFUSE_BLK2_RDATA5_REG	Returns data word 5 in eFuse BLOCK 2	0x3FF5A06C	RO
EFUSE_BLK2_RDATA6_REG	Returns data word 6 in eFuse BLOCK 2	0x3FF5A070	RO
EFUSE_BLK2_RDATA7_REG	Returns data word 7 in eFuse BLOCK 2	0x3FF5A074	RO
EFUSE_BLK3_RDATA0_REG	Returns data word 0 in eFuse BLOCK 3	0x3FF5A078	RO
EFUSE_BLK3_RDATA1_REG	Returns data word 1 in eFuse BLOCK 3	0x3FF5A07C	RO
EFUSE_BLK3_RDATA2_REG	Returns data word 2 in eFuse BLOCK 3	0x3FF5A080	RO
EFUSE_BLK3_RDATA3_REG	Returns data word 3 in eFuse BLOCK 3	0x3FF5A084	RO
EFUSE_BLK3_RDATA4_REG	Returns data word 4 in eFuse BLOCK 3	0x3FF5A088	RO
EFUSE_BLK3_RDATA5_REG	Returns data word 5 in eFuse BLOCK 3	0x3FF5A08C	RO
EFUSE_BLK3_RDATA6_REG	Returns data word 6 in eFuse BLOCK 3	0x3FF5A090	RO
EFUSE_BLK3_RDATA7_REG	Returns data word 7 in eFuse BLOCK 3	0x3FF5A094	RO
eFuse data write registers			
EFUSE_BLK0_RDATA0_REG	Writes data to word 0 in eFuse BLOCK 0	0x3FF5A000	RO
EFUSE_BLK0_RDATA1_REG	Writes data to word 1 in eFuse BLOCK 0	0x3FF5A004	RO
EFUSE_BLK0_RDATA2_REG	Writes data to word 2 in eFuse BLOCK 0	0x3FF5A008	RO
EFUSE_BLK0_RDATA3_REG	Writes data to word 3 in eFuse BLOCK 0	0x3FF5A00C	RO
EFUSE_BLK0_RDATA4_REG	Writes data to word 4 in eFuse BLOCK 0	0x3FF5A010	RO
EFUSE_BLK0_RDATA5_REG	Writes data to word 5 in eFuse BLOCK 0	0x3FF5A014	RO
EFUSE_BLK0_RDATA6_REG	Writes data to word 6 in eFuse BLOCK 0	0x3FF5A018	RO
EFUSE_BLK1_RDATA0_REG	Writes data to word 0 in eFuse BLOCK 1	0x3FF5A038	RO
EFUSE_BLK1_RDATA1_REG	Writes data to word 1 in eFuse BLOCK 1	0x3FF5A03C	RO
EFUSE_BLK1_RDATA2_REG	Writes data to word 2 in eFuse BLOCK 1	0x3FF5A040	RO
EFUSE_BLK1_RDATA3_REG	Writes data to word 3 in eFuse BLOCK 1	0x3FF5A044	RO
EFUSE_BLK1_RDATA4_REG	Writes data to word 4 in eFuse BLOCK 1	0x3FF5A048	RO
EFUSE_BLK1_RDATA5_REG	Writes data to word 5 in eFuse BLOCK 1	0x3FF5A04C	RO
EFUSE_BLK1_RDATA6_REG	Writes data to word 6 in eFuse BLOCK 1	0x3FF5A050	RO
EFUSE_BLK1_RDATA7_REG	Writes data to word 7 in eFuse BLOCK 1	0x3FF5A054	RO

Name	Description	Address	Access
EFUSE_BLK2_RDATA0_REG	Writes data to word 0 in eFuse BLOCK 2	0x3FF5A058	RO
EFUSE_BLK2_RDATA1_REG	Writes data to word 1 in eFuse BLOCK 2	0x3FF5A05C	RO
EFUSE_BLK2_RDATA2_REG	Writes data to word 2 in eFuse BLOCK 2	0x3FF5A060	RO
EFUSE_BLK2_RDATA3_REG	Writes data to word 3 in eFuse BLOCK 2	0x3FF5A064	RO
EFUSE_BLK2_RDATA4_REG	Writes data to word 4 in eFuse BLOCK 2	0x3FF5A068	RO
EFUSE_BLK2_RDATA5_REG	Writes data to word 5 in eFuse BLOCK 2	0x3FF5A06C	RO
EFUSE_BLK2_RDATA6_REG	Writes data to word 6 in eFuse BLOCK 2	0x3FF5A070	RO
EFUSE_BLK2_RDATA7_REG	Writes data to word 7 in eFuse BLOCK 2	0x3FF5A074	RO
EFUSE_BLK3_RDATA0_REG	Writes data to word 0 in eFuse BLOCK 3	0x3FF5A078	RO
EFUSE_BLK3_RDATA1_REG	Writes data to word 1 in eFuse BLOCK 3	0x3FF5A07C	RO
EFUSE_BLK3_RDATA2_REG	Writes data to word 2 in eFuse BLOCK 3	0x3FF5A080	RO
EFUSE_BLK3_RDATA3_REG	Writes data to word 3 in eFuse BLOCK 3	0x3FF5A084	RO
EFUSE_BLK3_RDATA4_REG	Writes data to word 4 in eFuse BLOCK 3	0x3FF5A088	RO
EFUSE_BLK3_RDATA5_REG	Writes data to word 5 in eFuse BLOCK 3	0x3FF5A08C	RO
EFUSE_BLK3_RDATA6_REG	Writes data to word 6 in eFuse BLOCK 3	0x3FF5A090	RO
EFUSE_BLK3_RDATA7_REG	Writes data to word 7 in eFuse BLOCK 3	0x3FF5A094	RO
Control registers			
EFUSE_CLK_REG	Timing configuration register	0x3FF5A0F8	R/W
EFUSE_CONF_REG	Opcode register	0x3FF5A0FC	R/W
EFUSE_CMD_REG	Read/write command register	0x3FF5A104	R/W
Interrupt registers			
EFUSE_INT_RAW_REG	Raw interrupt status	0x3FF5A108	RO
EFUSE_INT_ST_REG	Masked interrupt status	0x3FF5A10C	RO
EFUSE_INT_ENA_REG	Interrupt enable bits	0x3FF5A110	R/W
EFUSE_INT_CLR_REG	Interrupt clear bits	0x3FF5A114	WO
Misc registers			
EFUSE_DAC_CONF_REG	Efuse timing configuration	0x3FF5A118	R/W
EFUSE_DEC_STATUS_REG	Status of 3/4 coding scheme	0x3FF5A11C	RO

11.5 Registers

Register 11.1: EFUSE_BLK0_RDATA0_REG (0x000)

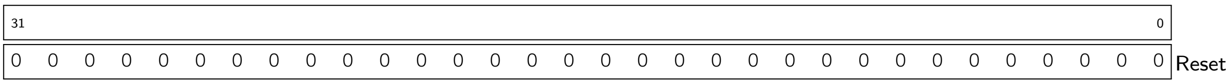


EFUSE_RD_FLASH_CRYPT_CNT This field returns the value of flash_crypt_cnt. (RO)

EFUSE_RD_EFUSE_RD_DIS This field returns the value of efuse_rd_disable. (RO)

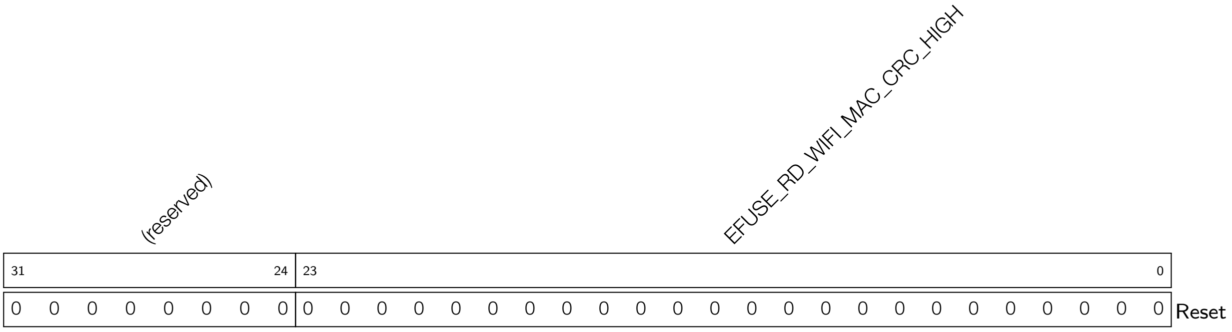
EFUSE_RD_EFUSE_WR_DIS This field returns the value of efuse_wr_disable. (RO)

Register 11.2: EFUSE_BLK0_RDATA1_REG (0x004)



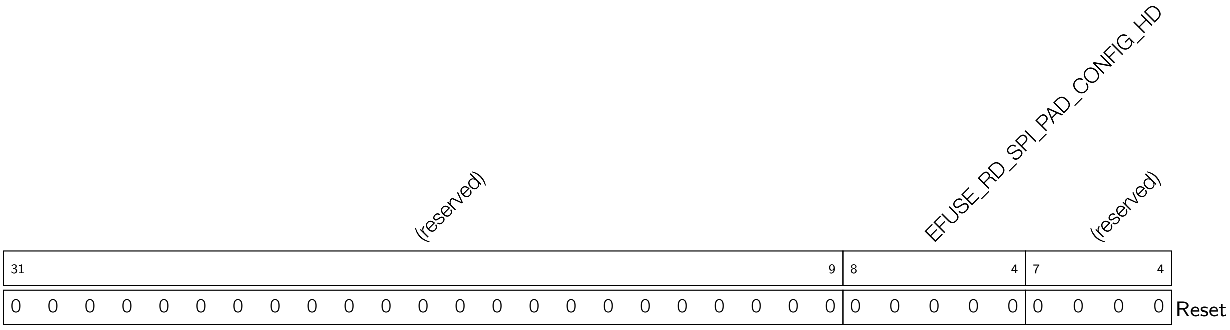
EFUSE_BLK0_RDATA1_REG This field returns the value of the lower 32 bits of WIFI_MAC_Address. (RO)

Register 11.3: EFUSE_BLK0_RDATA2_REG (0x008)



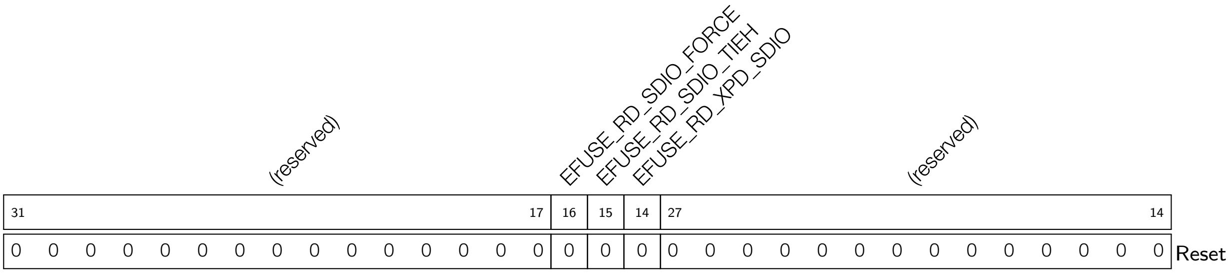
EFUSE_RD_WIFI_MAC_CRC_HIGH This field returns the value of the higher 24 bits of WIFI_MAC_Address. (RO)

Register 11.4: EFUSE_BLK0_RDATA3_REG (0x00c)



EFUSE_RD_SPI_PAD_CONFIG_HD This field returns the value of SPI_pad_config_hd. (RO)

Register 11.5: EFUSE_BLK0_RDATA4_REG (0x010)



EFUSE_RD_SDIO_FORCE This field returns the value of sdio_force. (RO)

EFUSE_RD_SDIO_TIEH This field returns the value of SDIO_TIEH. (RO)

EFUSE_RD_XPD_SDIO This field returns the value of XPD_SDIO_REG. (RO)

Register 11.6: EFUSE_BLK0_RDATA5_REG (0x014)

	EFUSE_RD_FLASH_CRYPT_CONFIG	(reserved)	EFUSE_RD_SPI_PAD_CONFIG_CS0	EFUSE_RD_SPI_PAD_CONFIG_D	EFUSE_RD_SPI_PAD_CONFIG_Q	EFUSE_RD_SPI_PAD_CONFIG_CLK
	31 28	27 20	19 15	14 10	9 5	4 0
Reset	0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0

EFUSE_RD_FLASH_CRYPT_CONFIG This field returns the value of flash_crypt_config. (RO)

EFUSE_RD_SPI_PAD_CONFIG_CS0 This field returns the value of SPI_pad_config_cs0. (RO)

EFUSE_RD_SPI_PAD_CONFIG_D This field returns the value of SPI_pad_config_d. (RO)

EFUSE_RD_SPI_PAD_CONFIG_Q This field returns the value of SPI_pad_config_q. (RO)

EFUSE_RD_SPI_PAD_CONFIG_CLK This field returns the value of SPI_pad_config_clk. (RO)

Register 11.7: EFUSE_BLK0_RDATA6_REG (0x018)

	(reserved)	EFUSE_RD_KEY_STATUS	EFUSE_RD_DISABLE_DL_CACHE	EFUSE_RD_DISABLE_DL_DECRYPT	EFUSE_RD_DISABLE_DL_ENCRYPT	EFUSE_RD_DISABLE_JTAG	EFUSE_RD_ABS_DONE_1	EFUSE_RD_ABS_DONE_0	(reserved)	EFUSE_RD_CONSOLE_DEBUG_DISABLE	EFUSE_RD_CODING_SCHEME
	31 11	10	9	8	7	6	5	4	3	2	1 0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0 0

EFUSE_RD_KEY_STATUS This field returns the value of key_status. (RO)

EFUSE_RD_DISABLE_DL_CACHE This field returns the value of download_dis_cache. (RO)

EFUSE_RD_DISABLE_DL_DECRYPT This field returns the value of download_dis_decrypt. (RO)

EFUSE_RD_DISABLE_DL_ENCRYPT This field returns the value of download_dis_encrypt. (RO)

EFUSE_RD_DISABLE_JTAG This field returns the value of JTAG_disable. (RO)

EFUSE_RD_ABS_DONE_1 This field returns the value of abstract_done_1. (RO)

EFUSE_RD_ABS_DONE_0 This field returns the value of abstract_done_0. (RO)

EFUSE_RD_CONSOLE_DEBUG_DISABLE This field returns the value of console_debug_disable. (RO)

EFUSE_RD_CODING_SCHEME This field returns the value of coding_scheme. (RO)

Register 11.8: EFUSE_BLK0_WDATA0_REG (0x01c)

(reserved)	EFUSE_FLASH_CRYPT_CNT	EFUSE_RD_DIS	EFUSE_WR_DIS	
31 28	27 20	19 16	15 0	
0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EFUSE_FLASH_CRYPT_CNT This field programs the value of flash_crypt_cnt. (R/W)

EFUSE_RD_DIS This field programs the value of efuse_rd_disable. (R/W)

EFUSE_WR_DIS This field programs the value of efuse_wr_disable. (R/W)

Register 11.9: EFUSE_BLK0_WDATA1_REG (0x020)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EFUSE_BLK0_WDATA1_REG This field programs the value of lower 32 bits of WIFI_MAC_Address. (R/W)

Register 11.10: EFUSE_BLK0_WDATA2_REG (0x024)

(reserved)	EFUSE_WIFI_MAC_CRC_HIGH	
31 24	23 0	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EFUSE_WIFI_MAC_CRC_HIGH This field programs the value of higher 24 bits of WIFI_MAC_Address. (R/W)

Register 11.11: EFUSE_BLK0_WDATA3_REG (0x028)

(reserved)	EFUSE_SPI_PAD_CONFIG_HD	(reserved)	
31 9	8 4	7 4	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0 0	Reset

EFUSE_SPI_PAD_CONFIG_HD This field programs the value of SPI_pad_config_hd. (R/W)

Register 11.12: EFUSE_BLK0_WDATA4_REG (0x02c)

(reserved)	EFUSE_SDIO_FORCE	EFUSE_SDIO_TIEH	EFUSE_XPD_SDIO	(reserved)	
31 17	16	15	14	27 14	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EFUSE_SDIO_FORCE This field programs the value of SDIO_TIEH. (R/W)

EFUSE_SDIO_TIEH This field programs the value of SDIO_TIEH. (R/W)

EFUSE_XPD_SDIO This field programs the value of XPD_SDIO_REG. (R/W)

Register 11.13: EFUSE_BLK0_WDATA5_REG (0x030)

EFUSE_FLASH_CRYPT_CONFIG	(reserved)	EFUSE_SPI_PAD_CONFIG_CS0	EFUSE_SPI_PAD_CONFIG_D	EFUSE_SPI_PAD_CONFIG_Q	EFUSE_SPI_PAD_CONFIG_CLK	
31 28	27 20	19 15	14 10	9 5	4 0	
0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	Reset

EFUSE_FLASH_CRYPT_CONFIG This field programs the value of flash_crypt_config. (R/W)

EFUSE_SPI_PAD_CONFIG_CS0 This field programs the value of SPI_pad_config_cs0. (R/W)

EFUSE_SPI_PAD_CONFIG_D This field programs the value of SPI_pad_config_d. (R/W)

EFUSE_SPI_PAD_CONFIG_Q This field programs the value of SPI_pad_config_q. (R/W)

EFUSE_SPI_PAD_CONFIG_CLK This field programs the value of SPI_pad_config_clk. (R/W)

Register 11.14: EFUSE_BLK0_WDATA6_REG (0x034)

(reserved)	EFUSE_KEY_STATUS	EFUSE_DISABLE_DL_CACHE	EFUSE_DISABLE_DL_DECRYPT	EFUSE_DISABLE_DL_ENCRYPT	EFUSE_DISABLE_JTAG	EFUSE_ABS_DONE_1	EFUSE_ABS_DONE_0	(reserved)	EFUSE_CONSOLE_DEBUG_DISABLE	EFUSE_CODING_SCHEME	
31 11	10	9	8	7	6	5	4	3	2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0 0	Reset

EFUSE_KEY_STATUS This field programs the value of key_status. (R/W)

EFUSE_DISABLE_DL_CACHE This field programs the value of download_dis_cache. (R/W)

EFUSE_DISABLE_DL_DECRYPT This field programs the value of download_dis_decrypt. (R/W)

EFUSE_DISABLE_DL_ENCRYPT This field programs the value of download_dis_encrypt. (R/W)

EFUSE_DISABLE_JTAG This field programs the value of JTAG_disable. (R/W)

EFUSE_ABS_DONE_1 This field programs the value of abstract_done_1. (R/W)

EFUSE_ABS_DONE_0 This field programs the value of abstract_done_0. (R/W)

EFUSE_CONSOLE_DEBUG_DISABLE This field programs the value of console_debug_disable. (R/W)

EFUSE_CODING_SCHEME This field programs the value of coding_scheme. (R/W)

Register 11.15: EFUSE_BLK1_RDATAn_REG (n: 0-7) (0x38+4*n)

31 0	
0x000000000	Reset

EFUSE_BLK1_RDATAn_REG This field returns the value of word *n* in BLOCK1. (RO)

Register 11.16: EFUSE_BLK2_RDATAn_REG (n: 0-7) (0x58+4*n)

31 0	
0x000000000	Reset

EFUSE_BLK2_RDATAn_REG This field returns the value of word *n* in BLOCK2. (RO)

Register 11.17: EFUSE_BLK3_RDATAn_REG (n: 0-7) (0x78+4*n)

31 0	
0x000000000	Reset

EFUSE_BLK3_RDATAn_REG This field returns the value of word *n* in BLOCK3. (RO)

Register 11.18: EFUSE_BLK1_WDATAn_REG (n: 0-7) (0x98+4*n)

31 0	
0x000000000	Reset

EFUSE_BLK1_WDATAn_REG This field programs the value of word *n* in of BLOCK1. (R/W)

Register 11.19: EFUSE_BLK2_WDATAn_REG (n: 0-7) (0xB8+4*n)

31 0	
0x000000000	Reset

EFUSE_BLK2_WDATAn_REG This field programs the value of word *n* in of BLOCK2. (R/W)

Register 11.20: EFUSE_BLK3_WDATAn_REG (n: 0-7) (0xD8+4*n)

31 0	
0x000000000	Reset

EFUSE_BLK3_WDATAn_REG This field programs the value of word *n* in of BLOCK3. (R/W)

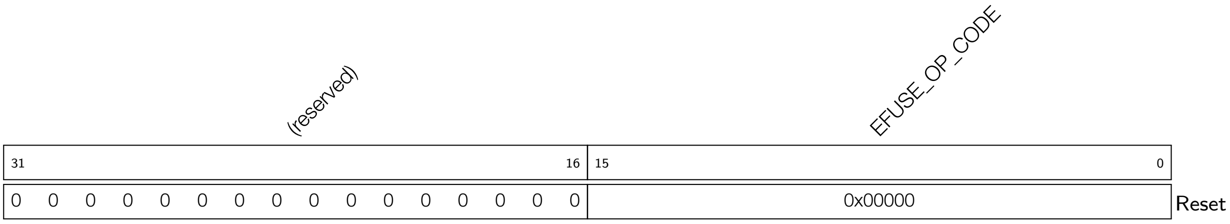
Register 11.21: EFUSE_CLK_REG (0x0f8)

(reserved)	EFUSE_CLK_SEL1	EFUSE_CLK_SEL0	
31 16	15 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x040	0x052	Reset

EFUSE_CLK_SEL1 eFuse clock configuration field. (R/W)

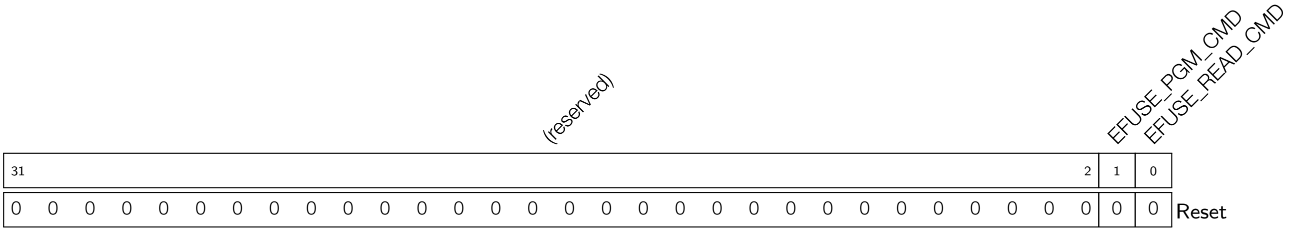
EFUSE_CLK_SEL0 eFuse clock configuration field. (R/W)

Register 11.22: EFUSE_CONF_REG (0x0fc)



EFUSE_OP_CODE eFuse operation code register. (R/W)

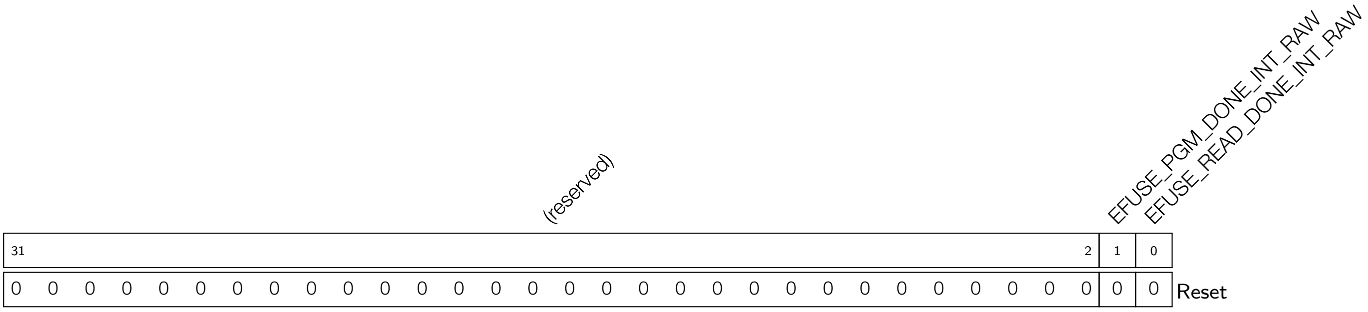
Register 11.23: EFUSE_CMD_REG (0x104)



EFUSE_PGM_CMD Set this to 1 to start a program operation. Reverts to 0 when the program operation is done. (R/W)

EFUSE_READ_CMD Set this to 1 to start a read operation. Reverts to 0 when the read operation is done. (R/W)

Register 11.24: EFUSE_INT_RAW_REG (0x108)



EFUSE_PGM_DONE_INT_RAW The raw interrupt status bit for the EFUSE_PGM_DONE_INT interrupt. (RO)

EFUSE_READ_DONE_INT_RAW The raw interrupt status bit for the EFUSE_READ_DONE_INT interrupt. (RO)

Register 11.25: EFUSE_INT_ST_REG (0x10c)

(reserved)	EFUSE_PGM_DONE_INT_ST	EFUSE_READ_DONE_INT_ST	
31 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

EFUSE_PGM_DONE_INT_ST The masked interrupt status bit for the EFUSE_PGM_DONE_INT interrupt. (RO)

EFUSE_READ_DONE_INT_ST The masked interrupt status bit for the EFUSE_READ_DONE_INT interrupt. (RO)

Register 11.26: EFUSE_INT_ENA_REG (0x110)

(reserved)	EFUSE_PGM_DONE_INT_ENA	EFUSE_READ_DONE_INT_ENA	
31 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

EFUSE_PGM_DONE_INT_ENA The interrupt enable bit for the EFUSE_PGM_DONE_INT interrupt. (R/W)

EFUSE_READ_DONE_INT_ENA The interrupt enable bit for the EFUSE_READ_DONE_INT interrupt. (R/W)

Register 11.27: EFUSE_INT_CLR_REG (0x114)

(reserved)	EFUSE_PGM_DONE_INT_CLR	EFUSE_READ_DONE_INT_CLR	
31 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

EFUSE_PGM_DONE_INT_CLR Set this bit to clear the EFUSE_PGM_DONE_INT interrupt. (WO)

EFUSE_READ_DONE_INT_CLR Set this bit to clear the EFUSE_READ_DONE_INT interrupt. (WO)

Register 11.28: EFUSE_DAC_CONF_REG (0x118)

(reserved)	EFUSE_DAC_CLK_DIV	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	40	Reset

EFUSE_DAC_CLK_DIV Efuse timing configuration register. (R/W)

Register 11.29: EFUSE_DEC_STATUS_REG (0x11c)

(reserved)	EFUSE_DEC_WARNINGS	
31 12	11 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0	Reset

EFUSE_DEC_WARNINGS If a bit is set in this register, it means some errors were corrected while decoding the 3/4 encoding scheme. (RO)

12. AES Accelerator

12.1 Introduction

The AES Accelerator speeds up AES operations significantly, compared to AES algorithms implemented solely in software. The AES Accelerator supports six algorithms of FIPS PUB 197, specifically AES-128, AES-192 and AES-256 encryption and decryption.

12.2 Features

- Supports AES-128 encryption and decryption
- Supports AES-192 encryption and decryption
- Supports AES-256 encryption and decryption
- Supports four variations of key endianness and four variations of text endianness

12.3 Functional Description

12.3.1 AES Algorithm Operations

The AES Accelerator supports six algorithms of FIPS PUB 197, specifically AES-128, AES-192 and AES-256 encryption and decryption. The AES_MODE_REG register can be configured to different values to enable different algorithm operations, as shown in Table 34.

Table 34: Operation Mode

AES_MODE_REG[2:0]	Operation
0	AES-128 Encryption
1	AES-192 Encryption
2	AES-256 Encryption
4	AES-128 Decryption
5	AES-192 Decryption
6	AES-256 Decryption

12.3.2 Key, Plaintext and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers. For AES-128 encryption/decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG. For AES-192 encryption/decryption, the 192-bit key is stored in AES_KEY_0_REG ~ AES_KEY_5_REG. For AES-256 encryption/decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

Plaintext and ciphertext is stored in the AES_TEXT_*m*_REG registers. There are four 32-bit registers. To enable AES-128/192/256 encryption, initialize the AES_TEXT_*m*_REG registers with plaintext before encryption. When encryption is finished, the AES Accelerator will store back the resulting ciphertext in the AES_TEXT_*m*_REG registers. To enable AES-128/192/256 decryption, initialize the AES_TEXT_*m*_REG registers with ciphertext before decryption. When decryption is finished, the AES Accelerator will store back the resulting plaintext in the AES_TEXT_*m*_REG registers.

12.3.3 Endianness

Key Endianness

Bit 0 and bit 1 in AES_ENDIAN_REG define the key endianness. For detailed information, please see Table 36, Table 37 and Table 38. w[0] ~ w[3] in Table 36, w[0] ~ w[5] in Table 37 and w[0] ~ w[7] in Table 38 are “the first Nk words of the expanded key” as specified in “5.2: Key Expansion” of FIPS PUB 197. “Column Bit” specifies the bytes in the word from w[0] to w[7]. The bytes of AES_KEY_*n*_REG comprise “the first Nk words of the expanded key”.

Text Endianness

Bit 2 and bit 3 in AES_ENDIAN_REG define the endianness of input text, while Bit 4 and Bit 5 define the endianness of output text. The input text refers to the plaintext in AES-128/192/256 encryption and the ciphertext in decryption. The output text refers to the ciphertext in AES-128/192/256 encryption and the plaintext in decryption. For details, please see Table 35. “State” in Table 35 is defined as that in “3.4: The State” of FIPS PUB 197: “The AES algorithm operations are performed on a two-dimensional array of bytes called the State”. The ciphertext or plaintexts stored in each byte of AES_TEXT_*m*_REG comprise the State.

Table 35: AES Text Endianness

AES_ENDIAN_REG[3]/[5]	AES_ENDIAN_REG[2]/[4]	Plaintext/Ciphertext					
0	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_3_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_0_REG[31:24]
			1	AES_TEXT_3_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_0_REG[23:16]
			2	AES_TEXT_3_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_0_REG[15:8]
			3	AES_TEXT_3_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_0_REG[7:0]
0	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_3_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_0_REG[7:0]
			1	AES_TEXT_3_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_0_REG[15:8]
			2	AES_TEXT_3_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_0_REG[23:16]
			3	AES_TEXT_3_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_0_REG[31:24]
1	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_0_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_3_REG[31:24]
			1	AES_TEXT_0_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_3_REG[23:16]
			2	AES_TEXT_0_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_3_REG[15:8]
			3	AES_TEXT_0_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_3_REG[7:0]
1	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_0_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_3_REG[7:0]
			1	AES_TEXT_0_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_3_REG[15:8]
			2	AES_TEXT_0_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_3_REG[23:16]
			3	AES_TEXT_0_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_3_REG[31:24]

Table 36: AES-128 Key Endianness

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]
0	0	[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

Table 37: AES-192 Key Endianness

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]
0	0	[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]

Table 38: AES-256 Key Endianness

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7]
0	0	[31:24]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
		[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
		[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
		[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

12.3.4 Encryption and Decryption Operations

Single Operation

1. Initialize AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_*m*_REG and AES_ENDIAN_REG.
2. Write 1 to AES_START_REG.
3. Wait until AES_IDLE_REG reads 1.
4. Read results from AES_TEXT_*m*_REG.

Consecutive Operations

Every time an operation is completed, only AES_TEXT_*m*_REG is modified by the AES Accelerator. Initialization can, therefore, be simplified in a series of consecutive operations.

1. Update contents of AES_MODE_REG, AES_KEY_*n*_REG and AES_ENDIAN_REG, if required.
2. Load AES_TEXT_*m*_REG.
3. Write 1 to AES_START_REG.
4. Wait until AES_IDLE_REG reads 1.
5. Read results from AES_TEXT_*m*_REG.

12.3.5 Speed

The AES Accelerator requires 11 to 15 clock cycles to encrypt a message block, and 21 or 22 clock cycles to decrypt a message block.

12.4 Register Summary

Name	Description	Address	Access
Configuration registers			
AES_MODE_REG	Mode of operation of the AES Accelerator	0x3FF01008	R/W
AES_ENDIAN_REG	Endianness configuration register	0x3FF01040	R/W
Key registers			
AES_KEY_0_REG	AES key material register 0	0x3FF01010	R/W
AES_KEY_1_REG	AES key material register 1	0x3FF01014	R/W
AES_KEY_2_REG	AES key material register 2	0x3FF01018	R/W
AES_KEY_3_REG	AES key material register 3	0x3FF0101C	R/W
AES_KEY_4_REG	AES key material register 4	0x3FF01020	R/W
AES_KEY_5_REG	AES key material register 5	0x3FF01024	R/W
AES_KEY_6_REG	AES key material register 6	0x3FF01028	R/W
AES_KEY_7_REG	AES key material register 7	0x3FF0102C	R/W
Encrypted/decrypted data registers			
AES_TEXT_0_REG	AES encrypted/decrypted data register 0	0x3FF01030	R/W
AES_TEXT_1_REG	AES encrypted/decrypted data register 1	0x3FF01034	R/W
AES_TEXT_2_REG	AES encrypted/decrypted data register 2	0x3FF01038	R/W
AES_TEXT_3_REG	AES encrypted/decrypted data register 3	0x3FF0103C	R/W
Control/status registers			

Name	Description	Address	Access
AES_START_REG	AES operation start control register	0x3FF01000	WO
AES_IDLE_REG	AES idle status register	0x3FF01004	RO

12.5 Registers

Register 12.1: AES_START_REG (0x000)

(reserved)	AES_START	
31 1	0	
0x00000000	x	Reset

AES_START Write 1 to start the AES operation. (WO)

Register 12.2: AES_IDLE_REG (0x004)

(reserved)	AES_IDLE	
31 1	0	
0x00000000	1	Reset

AES_IDLE AES Idle register. Reads ’zero’ while the AES Accelerator is busy processing; reads ’one’ otherwise. (RO)

Register 12.3: AES_MODE_REG (0x008)

(reserved)	AES_MODE	
31 3	2 0	
0x00000000	0	Reset

AES_MODE Selects the AES accelerator mode of operation. See Table 34 for details. (R/W)

Register 12.4: AES_KEY_*n*_REG (*n*: 0-7) (0x10+4**n*)

31 0	
0x000000000	Reset

AES_KEY_*n*_REG (*n*: 0-7) AES key material register. (R/W)

Register 12.5: AES_TEXT_*m*_REG (*m*: 0-3) (0x30+4**m*)

31 0	
0x000000000	Reset

AES_TEXT_*m*_REG (*m*: 0-3) Plaintext and ciphertext register. (R/W)

Register 12.6: AES_ENDIAN_REG (0x040)

(reserved)	AES_ENDIAN	
31 6	5 0	
0x0000000	1 1 1 1 1 1	Reset

AES_ENDIAN Endianness selection register. See Table 35 for details. (R/W)

13. SHA Accelerator

13.1 Introduction

The SHA Accelerator is included to speed up SHA hashing operations significantly, compared to SHA hashing algorithms implemented solely in software. The SHA Accelerator supports four algorithms of FIPS PUB 180-4, specifically SHA-1, SHA-256, SHA-384 and SHA-512.

13.2 Features

Hardware support for popular secure hashing algorithms:

- SHA-1
- SHA-256
- SHA-384
- SHA-512

13.3 Functional Description

13.3.1 Padding and Parsing the Message

The SHA Accelerator can only accept one message block at a time. Software divides the message into blocks according to "5.2 Parsing the Message" in FIPS PUB 180-4 and writes one block to the SHA_TEXT_*n*_REG registers each time. For SHA-1 and SHA-256, software writes a 512-bit message block to SHA_TEXT_0_REG ~ SHA_TEXT_15_REG each time. For SHA-384 and SHA-512, software writes a 1024-bit message block to SHA_TEXT_0_REG ~ SHA_TEXT_31_REG each time.

The SHA Accelerator is unable to perform the padding operation of "5.1 Padding the Message" in FIPS PUB 180-4; Note that the user software is expected to pad the message before feeding it into the accelerator.

As described in "2.2.1: Parameters" in FIPS PUB 180-4, "$M_0^{(i)}$ is the leftmost word of message block i". $M_0^{(i)}$ is stored in SHA_TEXT_0_REG. In the same fashion, the SHA_TEXT_1_REG register stores the second left-most word of a message block$H_1^{(N)}$, etc.

13.3.2 Message Digest

When the hashing operation is finished, the message digest will be refreshed by SHA Accelerator and will be stored in SHA_TEXT_*n*_REG. SHA-1 produces a 160-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_4_REG. SHA-256 produces a 256-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_7_REG. SHA-384 produces a 384-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_11_REG. SHA-512 produces a 512-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_15_REG.

As described in "2.2.1 Parameters" in FIPS PUB 180-4, "$H^{(N)}$ is the final hash value, and is used to determine the message digest", while "$H_0^{(i)}$ is the leftmost word of hash value i", so the leftmost word $H_0^{(N)}$ in the message digest is stored in SHA_TEXT_0_REG. In the same fashion, the second leftmost word $H_1^{(N)}$ in the message digest is stored in SHA_TEXT_1_REG, etc.

13.3.3 Hash Operation

There is a set of control registers for SHA-1, SHA-256, SHA-384 and SHA-512, respectively; different hashing algorithms use different control registers.
SHA-1 uses SHA_SHA1_START_REG, SHA_SHA1_CONTINUE_REG, SHA_SHA1_LOAD_REG and SHA_SHA1_BUSY_REG.
SHA-256 uses SHA_SHA256_START_REG, SHA_SHA256_CONTINUE_REG, SHA_SHA256_LOAD_REG and SHA_SHA256_BUSY_REG. SHA-384 uses SHA_SHA384_START_REG, SHA_SHA384_CONTINUE_REG, SHA_SHA384_LOAD_REG and SHA_SHA384_BUSY_REG.
SHA-512 uses SHA_SHA512_START_REG, SHA_SHA512_CONTINUE_REG, SHA_SHA512_LOAD_REG and SHA_SHA512_BUSY_REG. The following steps describe the operation in a detailed manner.

1. Feed the accelerator with the first message block:
   (a) Use the first message block to initialize SHA_TEXT_*n*_REG.
   (b) Write 1 to SHA_*X*_START_REG.
   (c) Wait for SHA_*X*_BUSY_REG to read 0, indicating that the operation is completed.
2. Similarly, feed the accelerator with subsequent message blocks:
   (a) Initialize SHA_TEXT_*n*_REG using the subsequent message block.
   (b) Write 1 to SHA_*X*_CONTINUE_REG.
   (c) Wait for SHA_*X*_BUSY_REG to read 0, indicating that the operation is completed.
3. Get message digest:
   (a) Write 1 to SHA_*X*_LOAD_REG.
   (b) Wait for SHA_*X*_BUSY_REG to read 0, indicating that operation is completed.
   (c) Read message digest from SHA_TEXT_*n*_REG.

13.3.4 Speed

The SHA Accelerator requires 60 to 100 clock cycles to process a message block and 8 to 20 clock cycles to calculate the final digest.

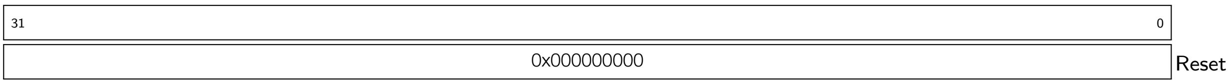
13.4 Register Summary

Name	Description	Address	Access
Encrypted/decrypted data registers			
SHA_TEXT_0_REG	SHA encrypted/decrypted data register 0	0x3FF03000	R/W
SHA_TEXT_1_REG	SHA encrypted/decrypted data register 1	0x3FF03004	R/W
SHA_TEXT_2_REG	SHA encrypted/decrypted data register 2	0x3FF03008	R/W
SHA_TEXT_3_REG	SHA encrypted/decrypted data register 3	0x3FF0300C	R/W
SHA_TEXT_4_REG	SHA encrypted/decrypted data register 4	0x3FF03010	R/W
SHA_TEXT_5_REG	SHA encrypted/decrypted data register 5	0x3FF03014	R/W
SHA_TEXT_6_REG	SHA encrypted/decrypted data register 6	0x3FF03018	R/W
SHA_TEXT_7_REG	SHA encrypted/decrypted data register 7	0x3FF0301C	R/W

Name	Description	Address	Access
SHA_TEXT_8_REG	SHA encrypted/decrypted data register 8	0x3FF03020	R/W
SHA_TEXT_9_REG	SHA encrypted/decrypted data register 9	0x3FF03024	R/W
SHA_TEXT_10_REG	SHA encrypted/decrypted data register 10	0x3FF03028	R/W
SHA_TEXT_11_REG	SHA encrypted/decrypted data register 11	0x3FF0302C	R/W
SHA_TEXT_12_REG	SHA encrypted/decrypted data register 12	0x3FF03030	R/W
SHA_TEXT_13_REG	SHA encrypted/decrypted data register 13	0x3FF03034	R/W
SHA_TEXT_14_REG	SHA encrypted/decrypted data register 14	0x3FF03038	R/W
SHA_TEXT_15_REG	SHA encrypted/decrypted data register 15	0x3FF0303C	R/W
SHA_TEXT_16_REG	SHA encrypted/decrypted data register 16	0x3FF03040	R/W
SHA_TEXT_17_REG	SHA encrypted/decrypted data register 17	0x3FF03044	R/W
SHA_TEXT_18_REG	SHA encrypted/decrypted data register 18	0x3FF03048	R/W
SHA_TEXT_19_REG	SHA encrypted/decrypted data register 19	0x3FF0304C	R/W
SHA_TEXT_20_REG	SHA encrypted/decrypted data register 20	0x3FF03050	R/W
SHA_TEXT_21_REG	SHA encrypted/decrypted data register 21	0x3FF03054	R/W
SHA_TEXT_22_REG	SHA encrypted/decrypted data register 22	0x3FF03058	R/W
SHA_TEXT_23_REG	SHA encrypted/decrypted data register 23	0x3FF0305C	R/W
SHA_TEXT_24_REG	SHA encrypted/decrypted data register 24	0x3FF03060	R/W
SHA_TEXT_25_REG	SHA encrypted/decrypted data register 25	0x3FF03064	R/W
SHA_TEXT_26_REG	SHA encrypted/decrypted data register 26	0x3FF03068	R/W
SHA_TEXT_27_REG	SHA encrypted/decrypted data register 27	0x3FF0306C	R/W
SHA_TEXT_28_REG	SHA encrypted/decrypted data register 28	0x3FF03070	R/W
SHA_TEXT_29_REG	SHA encrypted/decrypted data register 29	0x3FF03074	R/W
SHA_TEXT_30_REG	SHA encrypted/decrypted data register 30	0x3FF03078	R/W
SHA_TEXT_31_REG	SHA encrypted/decrypted data register 31	0x3FF0307C	R/W
Control/status registers			
SHA_SHA1_START_REG	Control register to initiate SHA1 operation	0x3FF03080	WO
SHA_SHA1_CONTINUE_REG	Control register to continue SHA1 operation	0x3FF03084	WO
SHA_SHA1_LOAD_REG	Control register to calculate the final SHA1 hash	0x3FF03088	WO
SHA_SHA1_BUSY_REG	Status register for SHA1 operation	0x3FF0308C	RO
SHA_SHA256_START_REG	Control register to initiate SHA256 operation	0x3FF03090	WO
SHA_SHA256_CONTINUE_REG	Control register to continue SHA256 operation	0x3FF03094	WO
SHA_SHA256_LOAD_REG	Control register to calculate the final SHA256 hash	0x3FF03098	WO
SHA_SHA256_BUSY_REG	Status register for SHA256 operation	0x3FF0309C	RO
SHA_SHA384_START_REG	Control register to initiate SHA384 operation	0x3FF030A0	WO
SHA_SHA384_CONTINUE_REG	Control register to continue SHA384 operation	0x3FF030A4	WO
SHA_SHA384_LOAD_REG	Control register to calculate the final SHA384 hash	0x3FF030A8	WO
SHA_SHA384_BUSY_REG	Status register for SHA384 operation	0x3FF030AC	RO
SHA_SHA512_START_REG	Control register to initiate SHA512 operation	0x3FF030B0	WO
SHA_SHA512_CONTINUE_REG	Control register to continue SHA512 operation	0x3FF030B4	WO
SHA_SHA512_LOAD_REG	Control register to calculate the final SHA512 hash	0x3FF030B8	WO
SHA_SHA512_BUSY_REG	Status register for SHA512 operation	0x3FF030BC	RO

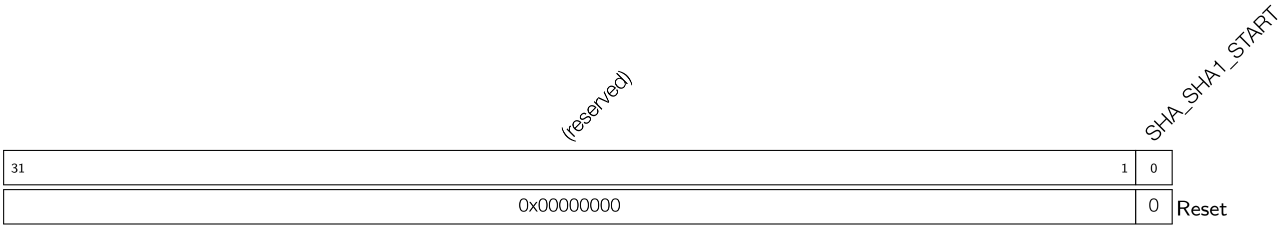
13.5 Registers

Register 13.1: SHA_TEXT_*n*_REG (*n*: 0-31) (0x0+4**n*)



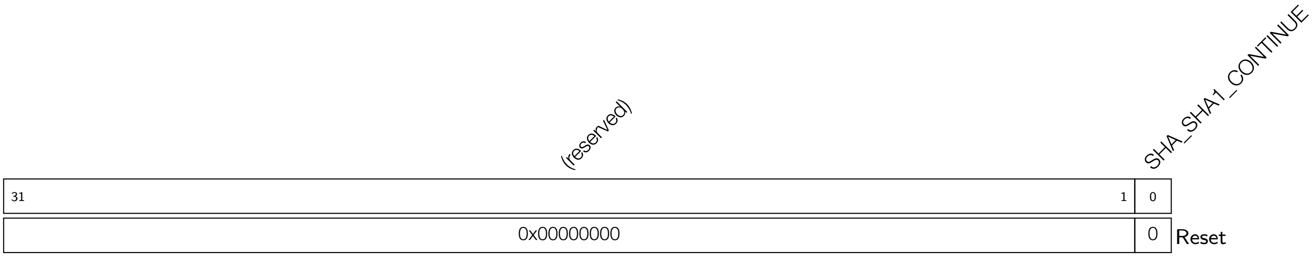
SHA_TEXT_*n*_REG (*n*: 0-31) SHA Message block and hash result register. (R/W)

Register 13.2: SHA_SHA1_START_REG (0x080)



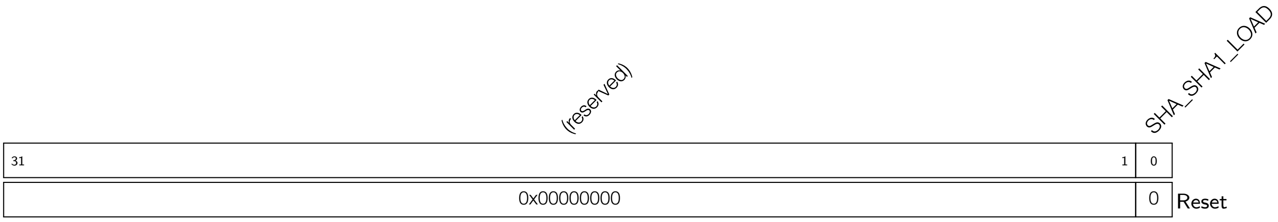
SHA_SHA1_START Write 1 to start an SHA-1 operation on the first message block. (WO)

Register 13.3: SHA_SHA1_CONTINUE_REG (0x084)



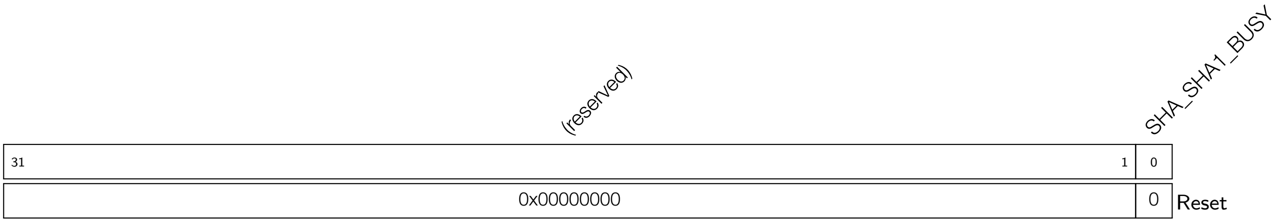
SHA_SHA1_CONTINUE Write 1 to continue the SHA-1 operation with subsequent blocks. (WO)

Register 13.4: SHA_SHA1_LOAD_REG (0x088)



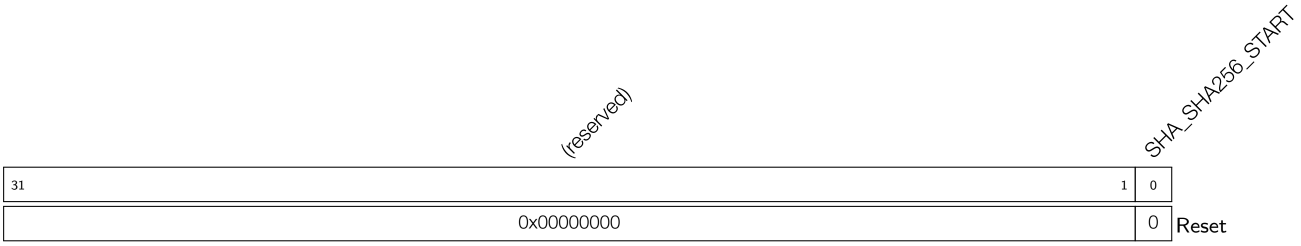
SHA_SHA1_LOAD Write 1 to finish the SHA-1 operation to calculate the final message hash. (WO)

Register 13.5: SHA_SHA1_BUSY_REG (0x08C)



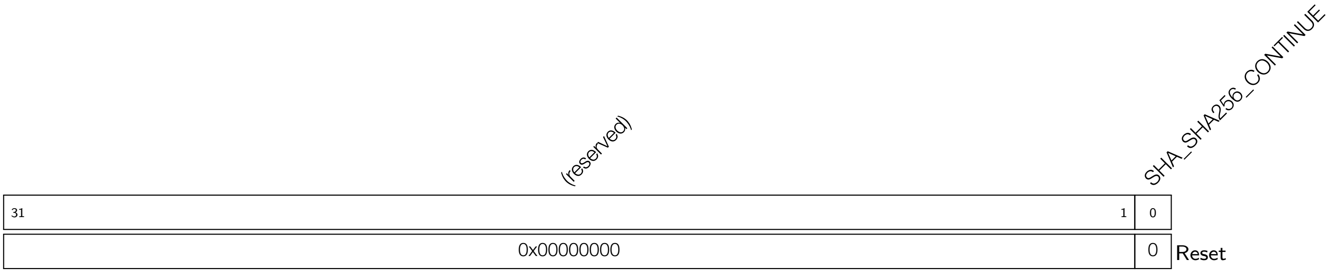
SHA_SHA1_BUSY SHA-1 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 13.6: SHA_SHA256_START_REG (0x090)



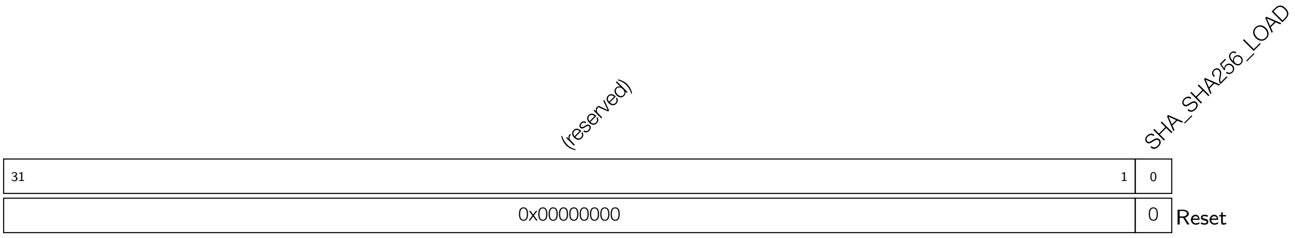
SHA_SHA256_START Write 1 to start an SHA-256 operation on the first message block. (WO)

Register 13.7: SHA_SHA256_CONTINUE_REG (0x094)



SHA_SHA256_CONTINUE Write 1 to continue the SHA-256 operation with subsequent blocks. (WO)

Register 13.8: SHA_SHA256_LOAD_REG (0x098)



SHA_SHA256_LOAD Write 1 to finish the SHA-256 operation to calculate the final message hash. (WO)

Register 13.9: SHA_SHA256_BUSY_REG (0x09C)

(reserved)	SHA_SHA256_BUSY	
31 1	0	
0x00000000	0	Reset

SHA_SHA256_BUSY SHA-256 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 13.10: SHA_SHA384_START_REG (0x0A0)

(reserved)	SHA_SHA384_START	
31 1	0	
0x00000000	0	Reset

SHA_SHA384_START Write 1 to start an SHA-384 operation on the first message block. (WO)

Register 13.11: SHA_SHA384_CONTINUE_REG (0x0A4)

(reserved)	SHA_SHA384_CONTINUE	
31 1	0	
0x00000000	0	Reset

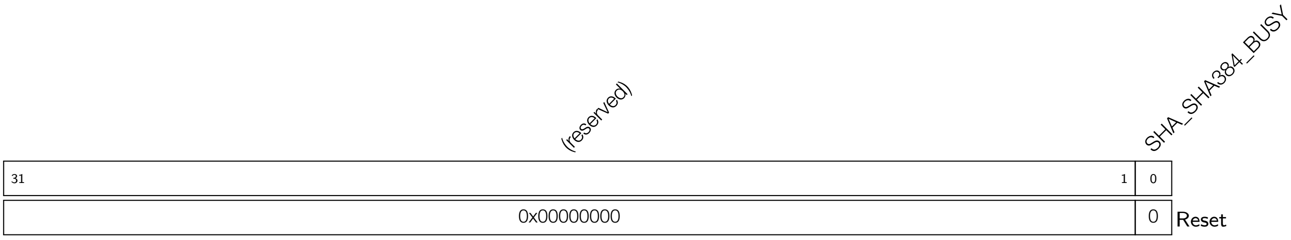
SHA_SHA384_CONTINUE Write 1 to continue the SHA-384 operation with subsequent blocks. (WO)

Register 13.12: SHA_SHA384_LOAD_REG (0x0A8)

(reserved)	SHA_SHA384_LOAD	
31 1	0	
0x00000000	0	Reset

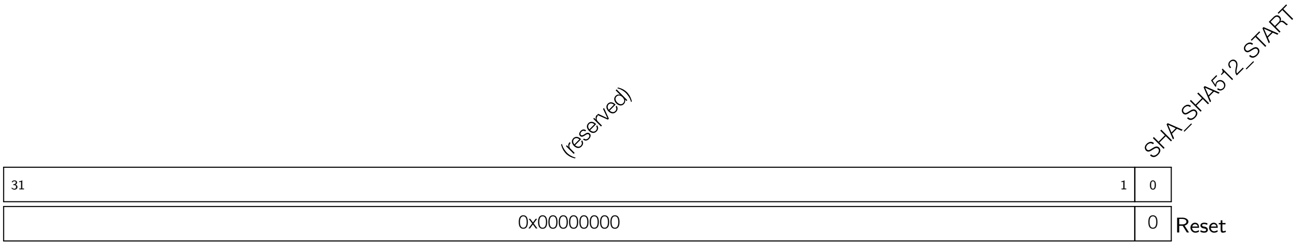
SHA_SHA384_LOAD Write 1 to finish the SHA-384 operation to calculate the final message hash. (WO)

Register 13.13: SHA_SHA384_BUSY_REG (0x0AC)



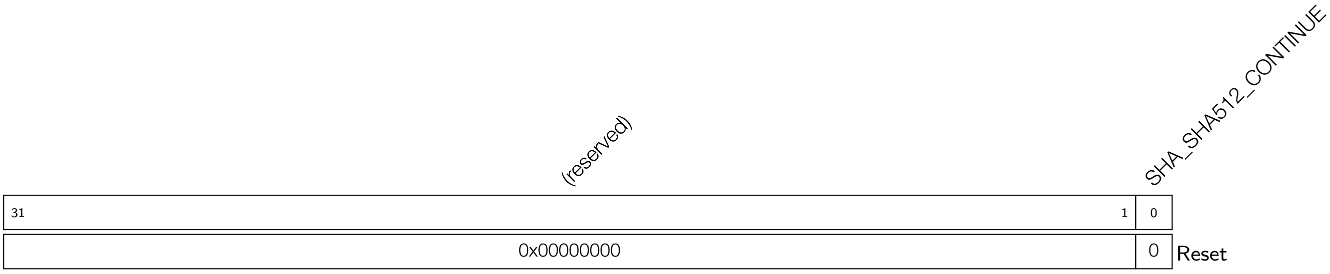
SHA_SHA384_BUSY SHA-384 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 13.14: SHA_SHA512_START_REG (0x0B0)



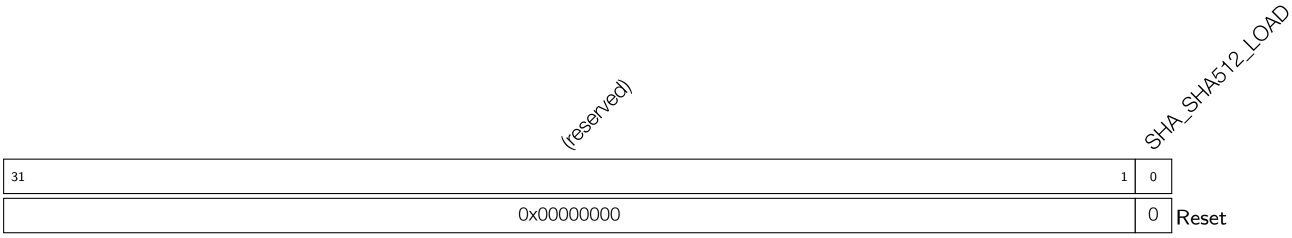
SHA_SHA512_START Write 1 to start an SHA-512 operation on the first message block. (WO)

Register 13.15: SHA_SHA512_CONTINUE_REG (0x0B4)



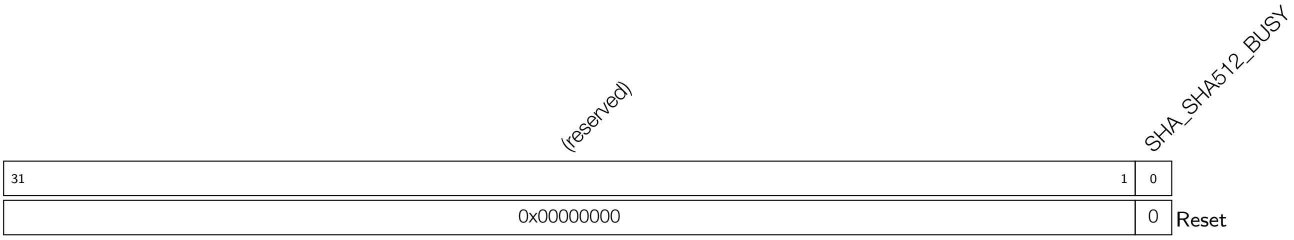
SHA_SHA512_CONTINUE Write 1 to continue the SHA-512 operation with subsequent blocks. (WO)

Register 13.16: SHA_SHA512_LOAD_REG (0x0B8)



SHA_SHA512_LOAD Write 1 to finish the SHA-512 operation to calculate the final message hash. (WO)

Register 13.17: SHA_SHA512_BUSY_REG (0x0BC)



SHA_SHA512_BUSY SHA-512 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

14. RSA Accelerator

14.1 Introduction

The RSA Accelerator provides hardware support for multiple precision arithmetic operations used in RSA asymmetric cipher algorithms.

Sometimes, multiple precision arithmetic is also called ”bignum arithmetic”, ”bigint arithmetic” or ”arbitrary precision arithmetic”.

14.2 Features

- Support for large-number modular exponentiation
- Support for large-number modular multiplication
- Support for large-number multiplication
- Support for various lengths of operands

14.3 Functional Description

14.3.1 Initialization

The RSA Accelerator is activated by enabling the corresponding peripheral clock, and by clearing the DPORT_RSA_PD bit in the DPORT_RSA_PD_CTRL_REG register. This releases the RSA Accelerator from reset.

When the RSA Accelerator is released from reset, the register RSA_CLEAN_REG reads 0 and an initialization process begins. Hardware initializes the four memory blocks by setting them to 0. After initialization is complete, RSA_CLEAN_REG reads 1. For this reason, software should query RSA_CLEAN_REG after being released from reset, and before writing to any RSA Accelerator memory blocks or registers for the first time.

14.3.2 Large Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The operation is based on Montgomery multiplication. Aside from the arguments X, Y, and M, two additional ones are needed — $\bar{r}$ and M'. These arguments are calculated in advance by software.

The RSA Accelerator supports operand lengths of $N \in \{512, 1024, 1536, 2048, 2560, 3072, 3584, 4096\}$ bits. The bit length of arguments Z, X, Y, M, and $\bar{r}$ can be any one from the N set, but all numbers in a calculation must be of the same length. The bit length of M' is always 32.

To represent the numbers used as operands, define a base-b positional notation, as follows:

$$b = 2^{32}$$

In this notation, each number is represented by a sequence of base-b digits, where each base-b digit is a 32-bit word. Representing an N-bit number requires n base-b digits (all of the possible N lengths are multiples of 32).

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the n values in Z_{n-1} ~ Z_0, X_{n-1} ~ X_0, Y_{n-1} ~ Y_0, M_{n-1} ~ M_0, $\overline{r}_{n-1}$ ~ $\overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define

$$R = b^n$$

then, we can calculate the additional arguments, as follows:

$$\overline{r} = R^2 \bmod M \tag{1}$$

$$\begin{cases} M'' \times M + 1 = R \times R^{-1} \\ M' = M'' \bmod b \end{cases} \tag{2}$$

(Equation 2 is written in a form suitable for calculations using the extended binary GCD algorithm.)

Software can implement large-number modular exponentiations in the following order:

1. Write $(\frac{N}{512} - 1)$ to RSA_MODEXP_MODE_REG.
2. Write X_i, Y_i, M_i and $\overline{r}_i$ ($i \in [0, n) \cap \mathbb{N}$) to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
3. Write M' to RSA_M_PRIME_REG.
4. Write 1 to RSA_MODEXP_START_REG.
5. Wait for the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
6. Read the result Z_i ($i \in [0, n) \cap \mathbb{N}$) from RSA_Z_MEM.
7. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, the RSA_MODEXP_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG will not have changed. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM will have been overwritten. In order to perform another operation, refresh the registers and memory blocks, as required.

14.3.3 Large Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y$ mod M. This operation is based on Montgomery multiplication. The same values $\overline{r}$ and M' are derived by software using the formulas 1 and 2 shown above.

The RSA Accelerator supports large-number modular multiplication with eight different operand lengths, which are the same as in the large-number modular exponentiation. The operation is performed by a combination of software and hardware. The software performs two hardware operations in sequence.

The software process is as follows:

1. Write $(\frac{N}{512} - 1)$ to RSA_MULT_MODE_REG.
2. Write X_i, M_i and $\overline{r}_i$ ($i \in [0, n) \cap \mathbb{N}$) to registers RSA_X_MEM, RSA_M_MEM and RSA_Z_MEM. Write data to each memory block only according to the length of the number. Data beyond this length are ignored.
3. Write M' to RSA_M_PRIME_REG.
4. Write 1 to RSA_MULT_START_REG.
5. Wait for the first round of the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
6. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.
7. Write Y_i ($i \in [0, n) \cap \mathbb{N}$) to RSA_X_MEM.

   Users need to write to the memory block only according to the length of the number. Data beyond this length are ignored.
8. Write 1 to RSA_MULT_START_REG.
9. Wait for the second round of the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
10. Read the result Z_i ($i \in [0, n) \cap \mathbb{N}$) from RSA_Z_MEM.
11. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, the RSA_MULT_MODE_REG register, and memory blocks RSA_M_MEM and RSA_M_PRIME_REG remain unchanged. Users do not need to refresh these registers or memory blocks if the values remain the same.

14.3.4 Large Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of Z is twice that of X and Y. Therefore, the RSA Accelerator supports large-number multiplication with only four operand lengths of $N \in \{512, 1024, 1536, 2048\}$ bits. The length $\hat{N}$ of the result Z is $2 \times N$ bits.

Operands X and Y need to be extended to form arguments $\hat{X}$ and $\hat{Y}$ which have the same length ($\hat{N}$ bits) as the result Z. X is left-extended and Y is right-extended, and defined as follows:

$$n = \frac{N}{32}$$

$$\hat{N} = 2 \times N$$

$$\hat{n} = \frac{\hat{N}}{32} = 2n$$

$$\hat{X} = (\hat{X}_{\hat{n}-1}\hat{X}_{\hat{n}-2}\cdots\hat{X}_0)_b = (\underbrace{00\cdots0}_{n}X)_b = (\underbrace{00\cdots0}_{n}X_{n-1}X_{n-2}\cdots X_0)_b$$

$$\hat{Y} = (\hat{Y}_{\hat{n}-1}\hat{Y}_{\hat{n}-2}\cdots\hat{Y}_0)_b = (Y\underbrace{00\cdots0}_{n})_b = (Y_{n-1}Y_{n-2}\cdots Y_0\underbrace{00\cdots0}_{n})_b$$

Software performs the operation in the following order:

1. Write $(\frac{\hat{N}}{512} - 1 + 8)$ to RSA_MULT_MODE_REG.
2. Write $\hat{X}_i$ and $\hat{Y}_i$ ($i \in [0, \hat{n}) \cap \mathbb{N}$) to RSA_X_MEM and RSA_Z_MEM, respectively.

   Write the valid data into each number's memory block, according to their lengths. Values beyond this length are ignored. Half of the base-b positional notations written to the memory are zero (using the derivations shown above). These zero values are indispensable.
3. Write 1 to RSA_MULT_START_REG.
4. Wait for the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
5. Read the result Z_i ($i \in [0, \hat{n}) \cap \mathbb{N}$) from RSA_Z_MEM.
6. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, only the RSA_MULT_MODE_REG register remains unmodified.

14.4 Register Summary

Name	Description	Address	Access
Configuration registers			
RSA_M_PRIME_REG	Register to store M'	0x3FF02800	R/W
Modular exponentiation registers			
RSA_MODEXP_MODE_REG	Modular exponentiation mode	0x3FF02804	R/W
RSA_MODEXP_START_REG	Start bit	0x3FF02808	WO
Modular multiplication registers			
RSA_MULT_MODE_REG	Modular multiplication mode	0x3FF0280C	R/W
RSA_MULT_START_REG	Start bit	0x3FF02810	WO
Misc registers			
RSA_INTERRUPT_REG	RSA interrupt register	0x3FF02814	R/W
RSA_CLEAN_REG	RSA clean register	0x3FF02818	RO

14.5 Registers

Register 14.1: RSA_M_PRIME_REG (0x800)

31 0	
0x000000000	Reset

RSA_M_PRIME_REG This register contains M’. (R/W)

Register 14.2: RSA_MODEXP_MODE_REG (0x804)

(reserved)	RSA_MODEXP_MODE	
31 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	Reset

RSA_MODEXP_MODE This register contains the mode of modular exponentiation. (R/W)

Register 14.3: RSA_MODEXP_START_REG (0x808)

(reserved)	RSA_MODEXP_START	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

RSA_MODEXP_START Write 1 to start modular exponentiation. (WO)

Register 14.4: RSA_MULT_MODE_REG (0x80C)

(reserved)	RSA_MULT_MODE	
31 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0	Reset

RSA_MULT_MODE This register contains the mode of modular multiplication and multiplication. (R/W)

Register 14.5: RSA_MULT_START_REG (0x810)

(reserved)	RSA_MULT_START	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

RSA_MULT_START Write 1 to start modular multiplication or multiplication. (WO)

Register 14.6: RSA_INTERRUPT_REG (0x814)

(reserved)	RSA_INTERRUPT	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

RSA_INTERRUPT RSA interrupt status register. Will read 1 once an operation has completed. (R/W)

Register 14.7: RSA_CLEAN_REG (0x818)

(reserved)	RSA_CLEAN	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

RSA_CLEAN This bit will read 1 once the memory initialization is completed. (RO)

15. Random Number Generator

15.1 Introduction

The ESP32 contains a true random number generator, whose values can be used as a basis for cryptographical operations, among other things.

15.2 Feature

It can generate true random numbers.

15.3 Functional Description

When used correctly, every 32-bit value the system reads from the RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the noise in the Wi-Fi/BT RF system. When Wi-Fi and BT are disabled, the random number generator will give out pseudo-random numbers.

When Wi-Fi or BT is enabled, the random number generator is fed two bits of entropy every APB clock cycle (normally 80 MHz). Thus, for the maximum amount of entropy, it is advisable to read the random register at a maximum rate of 5 MHz.

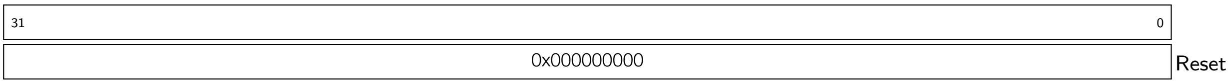
A data sample of 2 GB, read from the random number generator with Wi-Fi enabled and the random register read at 5 MHz, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

15.4 Register Summary

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x3FF75144	RO

15.5 Register

Register 15.1: RNG_DATA_REG (0x144)



RNG_DATA_REG Random number source. (RO)

16. PID/MPU/MMU

16.1 Introduction

Every peripheral and memory section in the ESP32 is accessed through either an MMU (Memory Management Unit) or an MPU (Memory Protection Unit). An MPU can allow or disallow the access of an application to a memory range or peripheral, depending on what kind of permission the OS has given to that particular application. An MMU can perform the same operation, as well as a virtual-to-physical memory address translation. This can be used to map an internal or external memory range to a certain virtual memory area. These mappings can be application-specific. Therefore, each application can be adjusted and have the memory configuration that is necessary for it to run properly. To differentiate between the OS and applications, there are eight Process Identifiers (or PIDs) that each application, or OS, can run. Furthermore, each application, or OS, is equipped with their own sets of mappings and rights.

16.2 Features

- Eight processes in each of the PRO_CPU and APP_CPU
- MPU/MMU management of on-chip memories, off-chip memories, and peripherals, based on process ID
- On-chip memory management by MPU/MMU
- Off-chip memory management by MMU
- Peripheral management by MPU

16.3 Functional Description

16.3.1 PID Controller

In the ESP32, a PID controller acts as an indicator that signals the MMU/MPU the owner PID of the code that is currently running. The intention is that the OS updates the PID in the PID controller every time it switches context to another application. The PID controller can detect interrupts and automatically switch PIDs to that of the OS, if so configured.

There are two peripheral PID controllers in the system, one for each of the two CPUs in the ESP32. Having a PID controller per CPU allows running different processes on different CPUs, if so desired.

16.3.2 MPU/MMU

The MPU and MMU manage on-chip memories, off-chip memories, and peripherals. To do this they are based on the process of accessing the peripheral or memory region. More specifically, when a code tries to access a MMU/MPU-protected memory region or peripheral, the MMU or MPU will receive the PID from the PID generator that is associated with the CPU on which the process is running.

For on-chip memory and peripherals, the decisions the MMU and MPU make are only based on this PID, whereas the specific CPU the code is running on is not taken into account. Subsequently, the MMU/MPU configuration for the internal memory and peripherals allows entries only for the eight different PIDs. In contrast, the MMU moderating access to the external memory takes not only the PID into account, but also the CPU the request is coming from. This means that MMUs have configuration options for every PID when running on the APP_CPU, as well as every PID when running on the PRO_CPU. While, in practice, accesses from both CPUs will be configured to have the same result for a specific process, doing so is not a hardware requirement.

The decision an MPU can make, based on this information, is to allow or deny a process to access the memory region or peripheral. An MMU has the same function, but additionally it redirects the virtual memory access, which the process acquired, into a physical memory access that can possibly reach out an entirely different physical memory region. This way, MMU-governed memory can be remapped on a process-by-process basis.

16.3.2.1 Embedded Memory

The on-chip memory is governed by fixed-function MPUs, configurable MPUs, and MMUs:

Table 43: MPU and MMU Structure for Internal Memory

Name	Size	Address range		Governed by
		From	To	
ROM0	384 KB	0x4000_0000	0x4005_FFFF	Static MPU
ROM1	64 KB	0x3FF9_0000	0x3FF9_FFFF	Static MPU
SRAM0	64 KB	0x4007_0000	0x4007_FFFF	Static MPU
	128 KB	0x4008_0000	0x4009_FFFF	SRAM0 MMU
SRAM1 (aliases)	128 KB	0x3FFE_0000	0x3FFF_FFFF	Static MPU
	128 KB	0x400A_0000	0x400B_FFFF	Static MPU
	32 KB	0x4000_0000	0x4000_7FFF	Static MPU
SRAM2	72 KB	0x3FFA_E000	0x3FFB_FFFF	Static MPU
	128 KB	0x3FFC_0000	0x3FFD_FFFF	SRAM2 MMU
RTC FAST (aliases)	8 KB	0x3FF8_0000	0x3FF8_1FFF	RTC FAST MPU
	8 KB	0x400C_0000	0x400C_1FFF	RTC FAST MPU
RTC SLOW	8 KB	0x5000_0000	0x5000_1FFF	RTC SLOW MPU

Static MPUs

ROM0, ROM1, the lower 64 KB of SRAM0, SRAM1 and the lower 72 KB of SRAM2 are governed by a static MPU. The behaviour of these MPUs are hardwired and cannot be configured by software. They moderate access to the memory region solely through the PID of the current process. When the PID of the process is 0 or 1, the memory can be read (and written when it is RAM) using the addresses specified in Table 43. When it is 2 ~ 7, the memory cannot be accessed.

RTC FAST & RTC SLOW MPU

The 8 KB RTC FAST Memory as well as the 8 KB of RTC SLOW Memory are governed by two configurable MPUs. The MPUs can be configured to allow or deny access to each individual PID, using the RTC_CNTL_RTC_PID_CONFIG_REG and DPORT_AHBLITE_MPU_TABLE_RTC_REG registers. Setting a bit in these registers will allow the corresponding PID to read or write from the memory; clearing the bit disallows access. Access for PID 0 and 1 to RTC SLOW memory cannot be configured and is always enabled. Table 44 and 45 define the bit-to-PID mappings of the registers.

Table 44: MPU for RTC FAST Memory

Size	Boundary address		Authority
	Low	High	PID RTC_CNTL_RTC_PID_CONFIG bit
8 KB	0x3FF8_0000	0x3FF8_1FFF	0 1 2 3 4 5 6 7
8 KB	0x400C_0000	0x400C_1FFF	0 1 2 3 4 5 6 7

Table 45: MPU for RTC SLOW Memory

Size	Boundary address		Authority	
	Low	High	PID = 0/1	PID DPORT_AHBLITE_MPU_TABLE_RTC_REG bit
8 KB	0x5000_0000	0x5000_1FFF	Read/Write	2 3 4 5 6 7 0 1 2 3 4 5

Register RTC_CNTL_RTC_PID_CONFIG_REG is part of the RTC peripheral and can only be modified by processes with a PID of 0; register DPORT_AHBLITE_MPU_TABLE_RTC_REG is a Dport register and can be changed by processes with a PID of 0 or 1.

SRAM0 and SRAM2 upper 128 KB MMUs

Both the upper 128 KB of SRAM0 and the upper 128 KB of SRAM2 are governed by an MMU. Not only can these MMUs allow or deny access to the memory they govern (just like the MPUs do), but they are also capable of translating the address a CPU reads from or writes to (which is a virtual address) to a possibly different address in memory (the physical address).

In order to accomplish this, the internal RAM MMUs divide the memory range they govern into 16 pages. The page size is configurable as 8 KB, 4 KB and 2 KB. When the page size is 8 KB, the 16 pages span the entire 128 KB memory region; when the page size is 4 KB or 2 KB, a non-MMU-covered region of 64 or 96 KB, respectively, will exist at the end of the memory space. Similar to the virtual and physical addresses, it is also possible to imagine the pages as having a virtual and physical component. The MMU can convert an address within a virtual page to an address within a physical page.

For PID 0 and 1, this mapping is 1-to-1, meaning that a read from or write to a certain virtual page will always be converted to a read from or write to the exact same physical page. This allows an operating system, running under PID 0 and/or 1, to always have access to the entire physical memory range.

For PID 2 to 7, however, every virtual page can be reconfigured, on a per-PID basis, to map to a different physical page. This way, reads and writes to an offset within a virtual page get translated into reads and writes to the same offset within a different physical page. This is illustrated in Figure 31: the CPU (running a process with a PID between 2 to 7) tries to access memory address 0x3FFC_2345. This address is within the virtual Page 1 memory region, at offset 0x0345. The MMU is instructed that for this particular PID, it should translate an access to virtual page 1 into physical Page 2. This causes the memory access to be redirected to the same offset as the virtual memory access, yet in Page 2, which results in the effective access of physical memory address 0x3FFC_**4**345. The page size in this example is 8 KB.

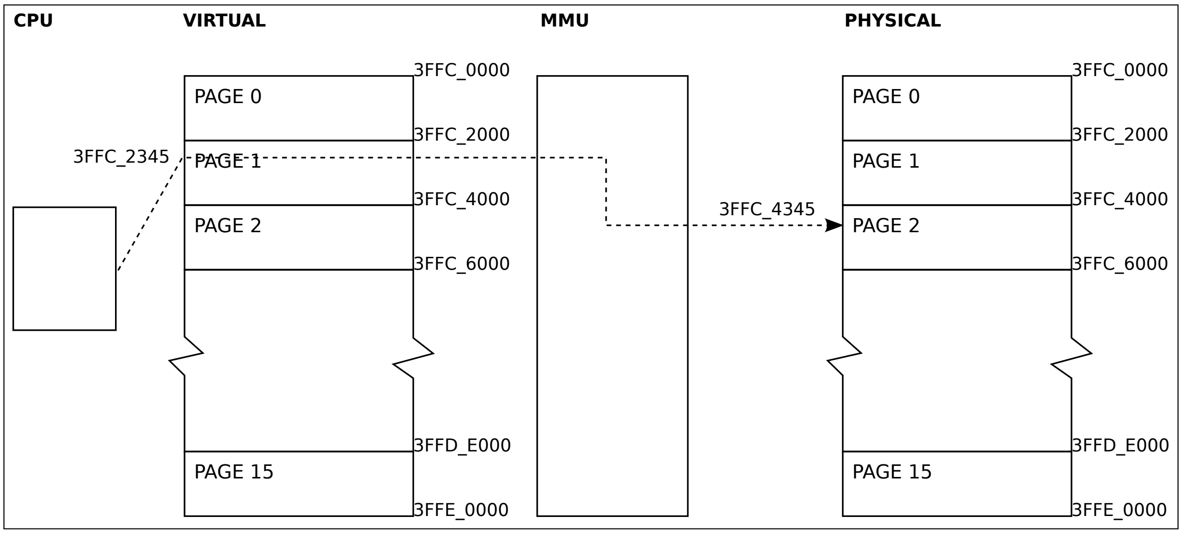


Figure 31: MMU Access Example

Table 46: Page Mode of MMU for the Remaining 128 KB of Internal SRAM0 and SRAM2

DPORT_IMMU_PAGE_MODE	DPORT_DMMU_PAGE_MODE	Page size
0	0	8 KB
1	1	4 KB
2	2	2 KB

Non-MMU Governed Memory

For the MMU-managed region of SRAM0 and SRAM2, the page size is configurable as 8 KB, 4 KB and 2 KB. The configuration is done by setting the DPORT_IMMU_PAGE_MODE (for SRAM0) and DPORT_DMMU_PAGE_MODE (for SRAM2) bits in registers DPORT_IMMU_PAGE_MODE_REG and DPORT_DMMU_PAGE_MODE_REG, as detailed in Table 46. Because the number of pages for either region is fixed at 16, the total amount of memory covered by these pages is 128 KB when 8 KB pages are selected, 64 KB when 4 KB pages are selected, and 32 KB when 2 KB pages are selected. This implies that for 8 KB pages, the entire MMU-managed range is used, but for the other page sizes there will be a part of the 128 KB memory that will not be governed by the MMU settings. Concretely, for a page size of 4 KB, these regions are 0x4009_0000 to 0x4009_FFFF and 0x3FFD_0000 to 0x3FFD_FFFF; for a page size of 2 KB, the regions are 0x4008_8000 to 0x4009_FFFF and 0x3FFC_8000 to 0x3FFD_FFFF. These ranges are readable and writable by processes with a PID of 0 or 1; processes with other PIDs cannot access this memory.

The layout of the pages in memory space is linear, namely, an SRAM0 MMU page n covers address space $0x40080000 + (pagesize * n)$ to $0x40080000 + (pagesize * (n + 1) - 1)$; similarly, an SRAM2 MMU page n covers $0x3FFC0000 + (pagesize * n)$ to $0x3FFC0000 + (pagesize * (n + 1) - 1)$. Tables 47 and 48 show the resulting addresses in full.

Table 47: Page Boundaries for SRAM0 MMU

Page	8 KB Pages		4 KB Pages		2 KB Pages	
	Bottom	Top	Bottom	Top	Bottom	Top
0	40080000	40081FFF	40080000	40080FFF	40080000	400807FF
1	40082000	40083FFF	40081000	40081FFF	40080800	40080FFF
2	40084000	40085FFF	40082000	40082FFF	40081000	400817FF
3	40086000	40087FFF	40083000	40083FFF	40081800	40081FFF
4	40088000	40089FFF	40084000	40084FFF	40082000	400827FF
5	4008A000	4008BFFF	40085000	40085FFF	40082800	40082FFF
6	4008C000	4008DFFF	40086000	40086FFF	40083000	400837FF
7	4008E000	4008FFFF	40087000	40087FFF	40083800	40083FFF
8	40090000	40091FFF	40088000	40088FFF	40084000	400847FF
9	40092000	40093FFF	40089000	40089FFF	40084800	40084FFF
10	40094000	40095FFF	4008A000	4008AFFF	40085000	400857FF
11	40096000	40097FFF	4008B000	4008BFFF	40085800	40085FFF
12	40098000	40099FFF	4008C000	4008CFFF	40086000	400867FF
13	4009A000	4009BFFF	4008D000	4008DFFF	40086800	40086FFF
14	4009C000	4009DFFF	4008E000	4008EFFF	40087000	400877FF
15	4009E000	4009FFFF	4008F000	4008FFFF	40087800	40087FFF
Rest	-	-	40090000	4009FFFF	4008800	4009FFFF

Table 48: Page Boundaries for SRAM2 MMU

Page	8 KB Pages		4 KB Pages		2 KB Pages	
	Bottom	Top	Bottom	Top	Bottom	Top
0	3FFC0000	3FFC1FFF	3FFC0000	3FFC0FFF	3FFC0000	3FFC07FF
1	3FFC2000	3FFC3FFF	3FFC1000	3FFC1FFF	3FFC0800	3FFC0FFF
2	3FFC4000	3FFC5FFF	3FFC2000	3FFC2FFF	3FFC1000	3FFC17FF
3	3FFC6000	3FFC7FFF	3FFC3000	3FFC3FFF	3FFC1800	3FFC1FFF
4	3FFC8000	3FFC9FFF	3FFC4000	3FFC4FFF	3FFC2000	3FFC27FF
5	3FFCA000	3FFCBFFF	3FFC5000	3FFC5FFF	3FFC2800	3FFC2FFF
6	3FFCC000	3FFCDFFF	3FFC6000	3FFC6FFF	3FFC3000	3FFC37FF
7	3FFCE000	3FFCFFFF	3FFC7000	3FFC7FFF	3FFC3800	3FFC3FFF
8	3FFD0000	3FFD1FFF	3FFC8000	3FFC8FFF	3FFC4000	3FFC47FF
9	3FFD2000	3FFD3FFF	3FFC9000	3FFC9FFF	3FFC4800	3FFC4FFF
10	3FFD4000	3FFD5FFF	3FFCA000	3FFCAFFF	3FFC5000	3FFC57FF
11	3FFD6000	3FFD7FFF	3FFCB000	3FFCBFFF	3FFC5800	3FFC5FFF
12	3FFD8000	3FFD9FFF	3FFCC000	3FFCCFFF	3FFC6000	3FFC67FF
13	3FFDA000	3FFDBFFF	3FFCD000	3FFCDFFF	3FFC6800	3FFC6FFF
14	3FFDC000	3FFDDFFF	3FFCE000	3FFCEFFF	3FFC7000	3FFC77FF
15	3FFDE000	3FFDFFFF	3FFCF000	3FFCFFFF	3FFC7800	3FFC7FFF
Rest	-	-	3FFD0000	3FFDFFFF	3FFC8000	3FFDFFFF

MMU Mapping

For each of the SRAM0 and SRAM2 MMUs, access rights and virtual to physical page mapping are done by a set of 16 registers. In contrast to most of the other MMUs, each register controls a physical page, not a virtual one. These registers control which of the PIDs have access to the physical memory, as well as which virtual page maps to this physical page. The bits in the register are described in Table 49. Keep in mind that these registers only govern accesses from processes with PID 2 to 7; PID 0 and 1 always have full read and write access to all pages and no virtual-to-physical mapping is done. In other words, if a process with a PID of 0 or 1 accesses virtual page *x*, the access will always go to physical page *x*, regardless of these register settings. These registers, as well as the page size selection registers DPORT_IMMU_PAGE_MODE_REG and DPORT_DMMU_PAGE_MODE_REG, are only writable from a process with PID 0 or 1.

Table 49: DPORT_DMMU_TABLE*n*_REG & DPORT_IMMU_TABLE*n*_REG

[6:4]	Access rights for PID 2 ~ 7
0	None of PIDs 2 ~ 7 have access.
1	All of PIDs 2 ~ 7 have access.
2	Only PID 2 has access.
3	Only PID 3 has access.
4	Only PID 4 has access.
5	Only PID 5 has access.
6	Only PID 6 has access.
7	Only PID 7 has access.

[3:0]	Address authority
0x00	Virtual page 0 accesses this physical page.
0x01	Virtual page 1 accesses this physical page.
0x02	Virtual page 2 accesses this physical page.
0x03	Virtual page 3 accesses this physical page.
0x04	Virtual page 4 accesses this physical page.
0x05	Virtual page 5 accesses this physical page.
0x06	Virtual page 6 accesses this physical page.
0x07	Virtual page 7 accesses this physical page.
0x08	Virtual page 8 accesses this physical page.
0x09	Virtual page 9 accesses this physical page.
0x10	Virtual page 10 accesses this physical page.
0x11	Virtual page 11 accesses this physical page.
0x12	Virtual page 12 accesses this physical page.
0x13	Virtual page 13 accesses this physical page.
0x14	Virtual page 14 accesses this physical page.
0x15	Virtual page 15 accesses this physical page.

Differences Between SRAM0 and SRAM2 MMU

The memory governed by the SRAM0 MMU is accessed through the processors I-bus, while the processor accesses the memory governed by the SRAM2 MMU through the D-bus. Thus, the normal envisioned use is for the code to be stored in the SRAM0 MMU pages and data in the MMU pages of SRAM2. In general, applications running under a PID of 2 to 7 are not expected to modify their own code, because for these PIDs access to the MMU pages of SRAM0 is read-only. These applications must, however, be able to modify their data section, so that they are allowed to read as well as write MMU pages located in SRAM2. As stated before, processes running under PID 0 or 1 always have full read-and-write access to both memory ranges.

DMA MPU

Applications may want to configure the DMA to send data straight from or to the peripherals they can control. With access to DMA, a malicious process may also be able to copy data from or to a region it cannot normally access. In order to be secure against that scenario, there is a DMA MPU which can be used to disallow DMA transfers from memory regions with sensitive data in them.

For each 8 KB region in the SRAM1 and SRAM2 regions, there is a bit in the DPORT_AHB_MPU_TABLE_*n*_REG registers which tells the MPU to either allow or disallow DMA access to this region. The DMA MPU uses only these bits to decide if a DMA transfer can be started; the PID of the process is not a factor. This means that when the OS wants to restrict its processes in a heterogenous fashion, it will need to re-load these registers with the values applicable to the process to be run on every context switch.

The register bits that govern access to the 8 KB regions are detailed in Table 50. When a register bit is set, DMA can read/write the corresponding 8 KB memory range. When the bit is cleared, access to that memory range is denied.

Table 50: MPU for DMA

Size	Boundary address		Authority	
	Low	High	Register	Bit
Internal SRAM 2				
8 KB	0x3FFA_E000	0x3FFA_FFFF	DPORT_AHB_MPU_TABLE_0_REG	0
8 KB	0x3FFB_0000	0x3FFB_1FFF	DPORT_AHB_MPU_TABLE_0_REG	1
8 KB	0x3FFB_2000	0x3FFB_3FFF	DPORT_AHB_MPU_TABLE_0_REG	2
8 KB	0x3FFB_4000	0x3FFB_5FFF	DPORT_AHB_MPU_TABLE_0_REG	3
8 KB	0x3FFB_6000	0x3FFB_7FFF	DPORT_AHB_MPU_TABLE_0_REG	4
8 KB	0x3FFB_8000	0x3FFB_9FFF	DPORT_AHB_MPU_TABLE_0_REG	5
8 KB	0x3FFB_A000	0x3FFB_BFFF	DPORT_AHB_MPU_TABLE_0_REG	6
8 KB	0x3FFB_C000	0x3FFB_DFFF	DPORT_AHB_MPU_TABLE_0_REG	7
8 KB	0x3FFB_E000	0x3FFB_FFFF	DPORT_AHB_MPU_TABLE_0_REG	8
8 KB	0x3FFC_0000	0x3FFC_1FFF	DPORT_AHB_MPU_TABLE_0_REG	9
8 KB	0x3FFC_2000	0x3FFC_3FFF	DPORT_AHB_MPU_TABLE_0_REG	10
8 KB	0x3FFC_4000	0x3FFC_5FFF	DPORT_AHB_MPU_TABLE_0_REG	11
8 KB	0x3FFC_6000	0x3FFC_7FFF	DPORT_AHB_MPU_TABLE_0_REG	12
8 KB	0x3FFC_8000	0x3FFC_9FFF	DPORT_AHB_MPU_TABLE_0_REG	13
8 KB	0x3FFC_A000	0x3FFC_BFFF	DPORT_AHB_MPU_TABLE_0_REG	14
8 KB	0x3FFC_C000	0x3FFC_DFFF	DPORT_AHB_MPU_TABLE_0_REG	15
8 KB	0x3FFC_E000	0x3FFC_FFFF	DPORT_AHB_MPU_TABLE_0_REG	16
8 KB	0x3FFD_0000	0x3FFD_1FFF	DPORT_AHB_MPU_TABLE_0_REG	17
8 KB	0x3FFD_2000	0x3FFD_3FFF	DPORT_AHB_MPU_TABLE_0_REG	18
8 KB	0x3FFD_4000	0x3FFD_5FFF	DPORT_AHB_MPU_TABLE_0_REG	19
8 KB	0x3FFD_6000	0x3FFD_7FFF	DPORT_AHB_MPU_TABLE_0_REG	20
8 KB	0x3FFD_8000	0x3FFD_9FFF	DPORT_AHB_MPU_TABLE_0_REG	21
8 KB	0x3FFD_A000	0x3FFD_BFFF	DPORT_AHB_MPU_TABLE_0_REG	22
8 KB	0x3FFD_C000	0x3FFD_DFFF	DPORT_AHB_MPU_TABLE_0_REG	23
8 KB	0x3FFD_E000	0x3FFD_FFFF	DPORT_AHB_MPU_TABLE_0_REG	24
Internal SRAM 1				
8 KB	0x3FFE_0000	0x3FFE_1FFF	DPORT_AHB_MPU_TABLE_0_REG	25
8 KB	0x3FFE_2000	0x3FFE_3FFF	DPORT_AHB_MPU_TABLE_0_REG	26
8 KB	0x3FFE_4000	0x3FFE_5FFF	DPORT_AHB_MPU_TABLE_0_REG	27
8 KB	0x3FFE_6000	0x3FFE_7FFF	DPORT_AHB_MPU_TABLE_0_REG	28

Size	Boundary address		Authority	
	Low	High	Register	Bit
8 KB	0x3FFE_8000	0x3FFE_9FFF	DPORT_AHB_MPU_TABLE_0_REG	29
8 KB	0x3FFE_A000	0x3FFE_BFFF	DPORT_AHB_MPU_TABLE_0_REG	30
8 KB	0x3FFE_C000	0x3FFE_DFFF	DPORT_AHB_MPU_TABLE_0_REG	31
8 KB	0x3FFE_E000	0x3FFE_FFFF	DPORT_AHB_MPU_TABLE_1_REG	0
8 KB	0x3FFF_0000	0x3FFF_1FFF	DPORT_AHB_MPU_TABLE_1_REG	1
8 KB	0x3FFF_2000	0x3FFF_3FFF	DPORT_AHB_MPU_TABLE_1_REG	2
8 KB	0x3FFF_4000	0x3FFF_5FFF	DPORT_AHB_MPU_TABLE_1_REG	3
8 KB	0x3FFF_6000	0x3FFF_7FFF	DPORT_AHB_MPU_TABLE_1_REG	4
8 KB	0x3FFF_8000	0x3FFF_9FFF	DPORT_AHB_MPU_TABLE_1_REG	5
8 KB	0x3FFF_A000	0x3FFF_BFFF	DPORT_AHB_MPU_TABLE_1_REG	6
8 KB	0x3FFF_C000	0x3FFF_DFFF	DPORT_AHB_MPU_TABLE_1_REG	7
8 KB	0x3FFF_E000	0x3FFF_FFFF	DPORT_AHB_MPU_TABLE_1_REG	8

Registers DPROT_AHB_MPU_TABLE_0_REG DPROT_AHB_MPU_TABLE_1_REG are located in the DPort address space. Only processes with a PID of 0 or 1 can modify these two registers.

16.3.2.2 External Memory

Accesses to the external flash and external SPI RAM are done through a cache and are also handled by an MMU. This Cache MMU can apply different mappings, depending on the PID of the process as well as the CPU the process is running on. The MMU does this in a way that is similar to the internal memory MMU, that is, for every page of virtual memory, it has a register detailing which physical page this virtual page should map to. There are differences between the MMUs governing the internal memory and the Cache MMU, though. First of all, the Cache MMU has a fixed page size (which is 64 KB for external flash and 32 KB for external RAM) and secondly, instead of specifying access rights in the MMU entries, the Cache MMU has explicit mapping tables for each PID and processor core. The MMU mapping configuration registers will be referred to as 'entries' in the rest of this chapter. These registers are only accessible from processes with a PID of 0 or 1; processes with a PID of 2 to 7 will have to delegate to one of the above-mentioned processes to change their MMU settings.

The MMU entries, as stated before, are used for mapping a virtual memory page access to a physical memory page access. The MMU controls five regions of virtual address space, detailed in Table 51. $VAddr_1$ to $VAddr_4$ are used for accessing external flash, whereas $VAddr_{RAM}$ is used for accessing external RAM. Note that $VAddr_4$ is a subset of $VAddr_0$.

Table 51: Virtual Address for External Memory

Name	Size	Boundary address		Page quantity
		Low	High	
$VAddr_0$	4 MB	0x3F40_0000	0x3F7F_FFFF	64
$VAddr_1$	4 MB	0x4000_0000	0x403F_FFFF	64*
$VAddr_2$	4 MB	0x4040_0000	0x407F_FFFF	64
$VAddr_3$	4 MB	0x4080_0000	0x40BF_FFFF	64
$VAddr_4$	1 MB	0x3F40_0000	0x3F4F_FFFF	16
$VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF	128

* The configuration entries for address range 0x4000_0000 ~ 0x403F_FFFF are implemented and documented as if it were a full 4 MB address range, but it is not accessible as such. Instead, the address range 0x4000_0000 ~ 0x400C_1FFF accesses on-chip memory. This means that some of the configuration entries for $VAddr_1$ will not be used.

External Flash

For flash, the relationships among entry numbers, virtual memory ranges, and PIDs are detailed in Tables 52 and 53, which for every memory region and PID combination specify the first MMU entry governing the mapping. This number refers to the MMU entry governing the very first page; the entire region is described by the amount of pages specified in the 'count' column.

These two tables are essentially the same, with the sole difference being that the APP_CPU entry numbers are 2048 higher than the corresponding PRO_CPU numbers. Note that memory regions $VAddr_0$ and $VAddr_1$ are only accessible using PID 0 and 1, while $VAddr_4$ can only be accessed by PID 2 ~ 7.

Table 52: MMU Entry Numbers for PRO_CPU

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	0	-	-	-	-	-	-
$VAddr_1$	64	64	-	-	-	-	-	-
$VAddr_2$	64	128	256	384	512	640	768	896
$VAddr_3$	64	192	320	448	576	704	832	960
$VAddr_4$	16	-	1056	1072	1088	1104	1120	1136

Table 53: MMU Entry Numbers for APP_CPU

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	2048	-	-	-	-	-	-
$VAddr_1$	64	2112	-	-	-	-	-	-
$VAddr_2$	64	2176	2304	2432	2560	2688	2816	2944
$VAddr_3$	64	2240	2368	2496	2624	2752	2880	3008
$VAddr_4$	16	-	3104	3120	3136	3152	3168	3184

As these tables show, virtual address $VAddr_1$ can only be used by processes with a PID of 0 or 1. There is a special mode to allow processes with a PID of 2 to 7 to read the External Flash via address $VAddr_1$. When the DPORT_PRO_SINGLE_IRAM_ENA bit of register DPORT_PRO_CACHE_CTRL_REG is 1, the MMU enters this special mode for PRO_CPU memory accesses. Similarily, when the DPORT_APP_SINGLE_IRAM_ENA bit of register DPORT_APP_CACHE_CTRL_REG is 1, the APP_CPU accesses memory using this special mode. In this mode, the process and virtual address page supported by each configuration entry of MMU are different. For details please see Table 54 and 55. As shown in these tables, in this special mode $VAddr_2$ and $VAddr_3$ cannot be used to access External Flash.

Table 54: MMU Entry Numbers for PRO_CPU (Special Mode)

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	0	-	-	-	-	-	-
$VAddr_1$	64	64	256	384	512	640	768	896
$VAddr_2$	64	-	-	-	-	-	-	-
$VAddr_3$	64	-	-	-	-	-	-	-
$VAddr_4$	16	-	1056	1072	1088	1104	1120	1136

Table 55: MMU Entry Numbers for APP_CPU (Special Mode)

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	2048	-	-	-	-	-	-
$VAddr_1$	64	2112	2304	2432	2560	2688	2816	2944
$VAddr_2$	64	-	-	-	-	-	-	-
$VAddr_3$	64	-	-	-	-	-	-	-
$VAddr_4$	16	-	3104	3120	3136	3152	3168	3184

Every configuration entry of MMU maps a virtual address page of a CPU process to a physical address page. An entry is 32 bits wide. Of these, bits 0~7 indicate the physical page the virtual page is mapped to. Bit 8 should be cleared to indicate that the MMU entry is valid; entries with this bit set will not map any physical address to the virtual address. Bits 10 to 32 are unused and should be written as zero. Because there are eight address bits in an MMU entry, and the page size for external flash is 64 KB, a maximum of 256 * 64 KB = 16 MB of external flash is supported.

Examples

Example 1. A PRO_CPU process, with a PID of 1, needs to read external flash address 0x07_2375 via virtual address 0x3F70_2375. The MMU is not in the special mode.

- According to Table 51, virtual address 0x3F70_2375 resides in the 0x30'th page of $VAddr_0$.
- According to Table 52, the MMU entry for $VAddr_0$ for PID 0/1 for the PRO_CPU starts at 0.
- The modified MMU entry is 0 + 0x30 = 0x30.
- Address 0x07_2375 resides in the 7'th 64 KB-sized page.
- MMU entry 0x30 needs to be set to 7 and marked as valid by setting the 8'th bit to 0. Thus, 0x007 is written to MMU entry 0x30.

Example 2. An APP_CPU process, with a PID of 4, needs to read external flash address 0x44_048C via virtual address 0x4044_048C. The MMU is not in special mode.

- According to Table 51, virtual address 0x4044_048C resides in the 0x4'th page of $VAddr_2$.
- According to Table 53, the MMU entry for $VAddr_2$ for PID 4 for the APP_CPU starts at 2560.
- The modified MMU entry is 2560 + 0x4 = 2564.
- Address 0x44_048C resides in the 0x44'th 64 KB-sized page.
- MMU entry 2564 needs to be set to 0x44 and marked as valid by setting the 8'th bit to 0. Thus, 0x044 is written to MMU entry 2564.

External RAM

Processes running on PRO_CPU and APP_CPU can read and write External SRAM via the Cache at virtual address range $VAddr_{RAM}$, which is 0x3F80_0000 ~ 0x3FBF_FFFF. As with the flash MMU, the address space and the physical memory are divided into pages. For the External RAM MMU, the page size is 32 KB and the MMU is able to map 256 physical pages into the virtual address space, allowing for 32 KB * 256 = 8 MB of physical external RAM to be mapped.

The mapping of virtual pages into this memory range depends on the mode this MMU is in: Low-High mode, Even-Odd mode, or Normal mode. In all cases, the DPORT_PRO_DRAM_HL bit and DPORT_PRO_DRAM_SPLIT bit in register DPORT_PRO_CACHE_CTRL_REG, the DPORT_APP_DRAM_HL bit and DPORT_APP_DRAM_SPLIT bit in register DPORT_APP_CACHE_CTRL_REG determine the virtual address mode for External SRAM. For details, please see Table 56. If a different mapping for the PRO_CPU and APP_CPU is required, the Normal Mode should be selected, as it is the only mode that can provide this. If it is allowable for the PRO_CPU and the APP_CPU to share the same mapping, using either High-Low or Even-Odd mode can give a speed gain when both CPUs access memory frequently.

In case the APP_CPU cache is disabled, which renders the region of 0x4007_8000 to 0x4007_FFFF usable as normal internal RAM, the usability of the various cache modes changes. Normal mode will allow PRO_CPU access to external RAM to keep functioning, but the APP_CPU will be unable to access the external RAM. High-Low mode allows both CPUs to use external RAM, but only for the 2 MB virtual memory addresses from 0x3F80_0000 to 0x3F9F_FFFF. It is not advised to use Even-Odd mode with the APP_CPU cache region disabled.

Table 56: Virtual Address Mode for External SRAM

Mode	DPORT_PRO_DRAM_HL DPORT_APP_DRAM_HL	DPORT_PRO_DRAM_SPLIT DPORT_APP_DRAM_SPLIT
Low-High	1	0
Even-Odd	0	1
Normal	0	0

In normal mode, the virtual-to-physical page mapping can be different for both CPUs. Page mappings for PRO_CPU are set using the MMU entries for ${}^{L}VAddr_{RAM}$, and page mappings for the APP_CPU can be configured using the MMU entries for ${}^{R}VAddr_{RAM}$. In this mode, all 128 pages of both ${}^{L}VAddr$ and ${}^{R}VAddr$ are fully used, allowing a maximum of 8 MB of memory to be mapped; 4 MB into PRO_CPU address space and a possibly different 4 MB into the APP_CPU address space, as can be seen in Table 57.

Table 57: Virtual Address for External SRAM (Normal Mode)

Virtual address	Size	PRO_CPU address	
		Low	High
$^{L}VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF
Virtual address	Size	APP_CPU address	
		Low	High
$^{R}VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF

In Low-High mode, both the PRO_CPU and the APP_CPU use the same mapping entries. In this mode $^{L}VAddr_{RAM}$ is used for the lower 2 MB of the virtual address space, while $^{R}VAddr_{RAM}$ is used for the upper 2 MB. This also means that the upper 64 MMU entries for $^{L}VAddr_{RAM}$, as well as the lower 64 entries for $^{R}VAddr_{RAM}$, are unused. Table 58 details these address ranges.

Table 58: Virtual Address for External SRAM (Low-High Mode)

Virtual address	Size	PRO_CPU/APP_CPU address	
		Low	High
$^{L}VAddr_{RAM}$	2 MB	0x3F80_0000	0x3F9F_FFFF
$^{R}VAddr_{RAM}$	2 MB	0x3FA0_0000	0x3FBF_FFFF

In Even-Odd memory, the VRAM is split into 32-byte chunks. The even chunks are resolved through the MMU entries for $^{L}VAddr_{RAM}$, the odd chunks through the entries for $^{R}VAddr_{RAM}$. Generally, the MMU entries for $^{L}VAddr_{RAM}$ and $^{R}VAddr_{RAM}$ are set to the same values, so that the virtual pages map to a contiguous region of physical memory. Table 59 details this mode.

Table 59: Virtual Address for External SRAM (Even-Odd Mode)

Virtual address	Size	PRO_CPU/APP_CPU address	
		Low	High
$^{L}VAddr_{RAM}$	32 Bytes	0x3F80_0000	0x3F80_001F
$^{R}VAddr_{RAM}$	32 Bytes	0x3F80_0020	0x3F80_003F
$^{L}VAddr_{RAM}$	32 Bytes	0x3F80_0040	0x3F80_005F
$^{R}VAddr_{RAM}$	32 Bytes	0x3F80_0060	0x3F80_007F
...			
$^{L}VAddr_{RAM}$	32 Bytes	0x3FBF_FFC0	0x3FBF_FFDF
$^{R}VAddr_{RAM}$	32 Bytes	0x3FBF_FFE0	0x3FBF_FFFF

The bit configuration of the External RAM MMU entries is the same as for the flash memory: the entries are 32-bit registers, with the lower nine bits being used. Bits 0~7 contain the physical page the entry should map its associate virtual page address to, while bit 8 is cleared when the entry is valid and set when it is not. Table 60 details the first MMU entry number for $^{L}VAddr_{RAM}$ and $^{R}VAddr_{RAM}$ for all PIDs.

Table 60: MMU Entry Numbers for External RAM

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$^{L}VAddr_{RAM}$	128	1152	1280	1408	1536	1664	1792	1920
$^{R}VAddr_{RAM}$	128	3200	3328	3456	3584	3712	3840	3968

Examples

Example 1. A PRO_CPU process, with a PID of 7, needs to read or write external RAM address 0x7F_A375 via virtual address 0x3FA7_2375. The MMU is in Low-High mode.

- According to Table 51, virtual address 0x3FA7_2375 resides in the 0x4E'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 58, virtual address 0x3FA7_2375 is governed by $^{R}VAddr_{RAM}$.
- According to Table 60, the MMU entry for $^{R}VAddr_{RAM}$ for PID 7 for the PRO_CPU starts at 3968.
- The modified MMU entry is 3968 + 0x4E = 4046.
- Address 0x7F_A375 resides in the 255'th 32 KB-sized page.
- MMU entry 4046 needs to be set to 255 and marked as valid by clearing the 8'th bit. Thus, 0x0FF is written to MMU entry 4046.

Example 2. An APP_CPU process, with a PID of 5, needs to read or write external RAM address 0x55_5805 up to 0x55_5823 starting at virtual address 0x3F85_5805. The MMU is in Even-Odd mode.

- According to Table 51, virtual address 0x3F85_5805 resides in the 0x0A'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 59, the range to be read/written spans both a 32-byte region in $^{R}VAddr_{RAM}$ and $^{L}VAddr_{RAM}$.
- According to Table 60, the MMU entry for $^{L}VAddr_{RAM}$ for PID 5 starts at 1664.
- According to Table 60, the MMU entry for $^{R}VAddr_{RAM}$ for PID 5 starts at 3712.
- The modified MMU entries are 1664 + 0x0A = 1674 and 3712 + 0x0A = 3722.
- The addresses 0x55_5805 to 0x55_5823 reside in the 0xAA'th 32 KB-sized page.
- MMU entries 1674 and 3722 need to be set to 0xAA and marked as valid by setting the 8'th bit to 0. Thus, 0x0AA is written to MMU entries 1674 and 3722. This mapping applies to both the PRO_CPU and the APP_CPU.

Example 3. A PRO_CPU process, with a PID of 1, and an APP_CPU process whose PID is also 1, need to read or write external RAM using virtual address 0x3F80_0876. The PRO_CPU needs this region to access physical address 0x10_0876, while the APP_CPU wants to access physical address 0x20_0876 through this virtual address. The MMU is in Normal mode.

- According to Table 51, virtual address 0x3F80_0876 resides in the 0'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 60, the MMU entry for PID 1 for the PRO_CPU starts at 1152.
- According to Table 60, the MMU entry for PID 1 for the APP_CPU starts at 3200.
- The MMU entries that are modified are 1152 + 0 = 1152 for the PRO_CPU and 3200 + 0 = 3200 for the APP_CPU.
- Address 0x10_0876 resides in the 0x20'th 32 KB-sized page.
- Address 0x20_0876 resides in the 0x40'th 32 KB-sized page.
- For the PRO_CPU, MMU entry 1152 needs to be set to 0x20 and marked as valid by clearing the 8'th bit. Thus, 0x020 is written to MMU entry 1152.

- For the APP_CPU, MMU entry 3200 needs to be set to 0x40 and marked as valid by clearing the 8'th bit. Thus, 0x040 is written to MMU entry 3200.
- Now, the PRO_CPU and the APP_CPU can access different physical memory regions through the same virtual address.

16.3.2.3 Peripheral

The Peripheral MPU manages the 41 peripheral modules. This MMU can be configured per peripheral to only allow access from a process with a certain PID. The registers to configure this are detailed in Table 61.

Table 61: MPU for Peripheral

Peripheral	Authority	
	PID = 0/1	PID = 2 ~ 7
DPort Register	Access	Forbidden
AES Accelerator	Access	Forbidden
RSA Accelerator	Access	Forbidden
SHA Accelerator	Access	Forbidden
Secure Boot	Access	Forbidden
Cache MMU Table	Access	Forbidden
PID Controller	Access	Forbidden
UART0	Access	DPORT_AHBLITE_MPU_TABLE_UART_REG
SPI1	Access	DPORT_AHBLITE_MPU_TABLE_SPI1_REG
SPI0	Access	DPORT_AHBLITE_MPU_TABLE_SPI0_REG
GPIO	Access	DPORT_AHBLITE_MPU_TABLE_GPIO_REG
RTC	Access	DPORT_AHBLITE_MPU_TABLE_RTC_REG
IO MUX	Access	DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG
SDIO Slave	Access	DPORT_AHBLITE_MPU_TABLE_HINF_REG
UDMA1	Access	DPORT_AHBLITE_MPU_TABLE_UHCI1_REG
I2S0	Access	DPORT_AHBLITE_MPU_TABLE_I2S0_REG
UART1	Access	DPORT_AHBLITE_MPU_TABLE_UART1_REG
I2C0	Access	DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG
UDMA0	Access	DPORT_AHBLITE_MPU_TABLE_UHCI0_REG
SDIO Slave	Access	DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG
RMT	Access	DPORT_AHBLITE_MPU_TABLE_RMT_REG
PCNT	Access	DPORT_AHBLITE_MPU_TABLE_PCNT_REG
SDIO Slave	Access	DPORT_AHBLITE_MPU_TABLE_SLC_REG
LED PWM	Access	DPORT_AHBLITE_MPU_TABLE_LEDC_REG
Efuse Controller	Access	DPORT_AHBLITE_MPU_TABLE_EFUSE_REG
Flash Encryption	Access	DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG
PWM0	Access	DPORT_AHBLITE_MPU_TABLE_PWM0_REG
TIMG0	Access	DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG
TIMG1	Access	DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG
SPI2	Access	DPORT_AHBLITE_MPU_TABLE_SPI2_REG
SPI3	Access	DPORT_AHBLITE_MPU_TABLE_SPI3_REG
SYSCON	Access	DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG

Peripheral	Authority	
	PID = 0/1	PID = 2 ~ 7
I2C1	Access	DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG
SDMMC	Access	DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG
EMAC	Access	DPORT_AHBLITE_MPU_TABLE_EMAC_REG
PWM1	Access	DPORT_AHBLITE_MPU_TABLE_PWM1_REG
I2S1	Access	DPORT_AHBLITE_MPU_TABLE_I2S1_REG
UART2	Access	DPORT_AHBLITE_MPU_TABLE_UART2_REG
PWM2	Access	DPORT_AHBLITE_MPU_TABLE_PWM2_REG
PWM3	Access	DPORT_AHBLITE_MPU_TABLE_PWM3_REG
RNG	Access	DPORT_AHBLITE_MPU_TABLE_PWR_REG

Each bit of register DPORT_AHBLITE_MPU_TABLE_X_REG determines whether each process can access the peripherals managed by the register. For details please see Table 62. When a bit of register DPORT_AHBLITE_MPU_TABLE_X_REG is 1, it means that a process with the corresponding PID can access the corresponding peripheral of the register. Otherwise, the process cannot access the corresponding peripheral.

Table 62: DPORT_AHBLITE_MPU_TABLE_X_REG

PID	2 3 4 5 6 7
DPORT_AHBLITE_MPU_TABLE_X_REG bit	0 1 2 3 4 5

All the DPORT_AHBLITE_MPU_TABLE_X_REG registers are in peripheral DPort Register. Only processes with PID 0/1 can modify these registers.